《数学机械化丛书》获国家基础研究发展规划项目“数学机械化方法及其在信息技术中的应用”与“数学机械化应用推广专项经费”资助

# 《数学机械化丛书》编委会

**主　编**　吴文俊

**副主编**　高小山

**编　委**　(以姓氏笔画为序)

万哲先　王东明　石　赫　冯果忱
刘卓军　齐东旭　李文林　李邦河
李洪波　杨　路　吴　可　吴文达
张景中　陈永川　周咸青　胡国定

数学机械化丛书　10

# 进 程 代 数

## ——对称与动作细化

王永祥　吴尽昭　蒋建民　著

科 学 出 版 社

北 京

# 内 容 简 介

并发系统常常在结构上展示出对称性，这种对称结构一般来说具有相同或相近的性质. 本书讨论了并发系统的进程代数语言及其事件结构模型中的对称性、对称约简、对称约简对动作细化的影响以及基于束动作变迁的偏序约简与应用，力图在结构层次上建立对建模语言和模型进行约简及细化的基本理论和方法，为高效机械化设计和分析并发系统服务.

本书可以供高年级大学生、研究生、教师和科研人员作为了解数学机械化基本思想与方法在形式化并发系统设计与分析中应用的参考书.

**图书在版编目(CIP)数据**

进程代数：对称与动作细化/王永祥，吴尽昭，蒋建民著. —北京：科学出版社，2007

(数学机械化丛书；10/吴文俊主编)

ISBN 978-7-03-018865-6

I.进… II. ①王… ②吴… ③蒋… III. 电子计算机-算法理论 IV. TP301.6

中国版本图书馆 CIP 数据核字(2007)第 055040 号

责任编辑：鄢德平 赵彦超 / 责任校对：赵桂芬

责任印制：赵德静 / 封面设计：陈 敬

科 学 出 版 社 出版

北京东黄城根北街 16 号

邮政编码：100717

http://www.sciencep.com

中国科学院印刷厂 印刷

科学出版社编务公司排版制作

科学出版社发行 各地新华书店经销

*

2007 年 6 月第 一 版 开本：B5 (720×1000)

2007 年 6 月第一次印刷 印张：6 3/4

印数：1—3 000 字数：108 000

**定价：25.00 元**

(如有印装质量问题，我社负责调换〈科印〉)

# 《数学机械化丛书》前言[①]

十六七世纪以来，人类历史上经历了一场史无前例的技术革命，出现了各种类型的机器，取代各种形式的体力劳动，使人类进入一个新时代．几百年后的今天，电子计算机已可开始有条件地代替一部分特定的脑力劳动，因而人类已面临另一场更宏伟的技术革命，处在又一个新时代的前夕．数学是一种典型的脑力劳动，它在这一场新的技术革命中，无疑将扮演一个重要的角色．为了了解数学在当前这场革命中所扮演的角色，就应对机器的作用，以及作为数学的脑力劳动的方式，进行一定的分析．

**1．什么是数学的机械化**

不论是机器代替体力劳动，或是计算机代替某种脑力劳动，其所以成为可能，关键在于所需代替的劳动已经“机械化”，也就是说已实现了刻板化或规格化．正因为割麦、刈草、纺纱、织布的动作已经是机械化刻板化了的，因而可据此造出割麦机、刈草机、纺纱机、织布机来．也正因为加减乘除开方等运算这一类脑力劳动，几千年来就已经是机械地刻板地进行的，才有可能使得 17 世纪的法国数学家 Pascal，利用齿轮传动造出了第一台机械计算机——加法机，并由 Leibniz 改进成为也能进行乘法的机器．数学问题的机械化，就要求在运算或证明过程中，每前进一步之后，都有一个确定的、必须选择的下一步，这样沿着一条有规律的、刻板的道路，一直达到结论．

在中小学数学的范围里，就有着不少已经机械化了的课题．除了四则、开方等运算外，解线性联立方程组就是一个很好的例子．在中学用的数学课本中，往往介绍解线性方程组的各种“消去法”，其求解过程是一个按一定程序进行的计算过程，也就是一种机械的、刻板的过程．根据这一过程编成程序，由电子计算机付诸实施，就可以不仅机器化而且达到自动化，在几分钟甚至几秒钟之内求出一个未知数多至上百个的线性方程组的解答来，这在手工计算几乎是不可能的．如果

① 20 世纪七八十年代之交，我尝试用计算机证明几何定理取得成功，由此提出了数学机械化的设想．先后在一些通俗报告与写作中，解释数学机械化的意义与前景，例如 1978 年发表于《自然辩证法通讯》的“数学机械化问题”以及 1980 年发表于《百科知识》的 “数学的机械化”．二文都重载于 1995 年由山东教育出版社出版的《吴文俊论数学机械化》一书．经过 20 多年众多学者的努力，数学机械化在各个方面都取得了丰富多彩的成就，并已出版了多种专著，汇集成现在的数学机械化丛书．现据 1980 年的《百科知识》的“数学的机械化”一文，稍加修改并作增补，以代丛书前言．

用手工计算, 即使是解只有三四个未知数的方程组, 也将是繁琐而令人厌烦的. 现代化的国防、经济建设中, 大量出现的例如网络一类的问题, 往往可归结为求解很多未知数的线性方程组. 这使得已经机械化了的线性方程解法在四个现代化中起着一种重要作用.

即使是不专门研究数学的人们, 也大都知道, 数学的脑力劳动有两种主要形式: 数值计算与定理证明(或许还应包括公式推导, 但这终究是次要的). 著名的数理逻辑学家美国洛克菲勒大学教授王浩先生在一篇有名的《向机械化数学前进》的文章中, 曾列举了这两种数学脑力劳动的若干不同之点. 我们可以简略而概括地把它们对比一下:

| 计算 | 证明 |
| --- | --- |
| 易 | 难 |
| 繁 | 简 |
| 刻板 | 灵活 |
| 枯燥 | 美妙 |

计算, 如已经提到过的加、减、乘、除、开方与解线性方程组, 其所以虽繁而易, 根本原因正在于它已经机械化. 而证明的巧而难, 是大家都深有体会的, 其根本原因也正在于它并没有机械化. 例如, 我们在中学初等几何定理的证明中, 就经常要依靠诸如直观、洞察、经验、以及其他一些模糊不清的原则, 去寻找捷径.

### 2. 从证明的机械化到机器证明

一个值得提出的问题是: 定理的证明是不是也能像计算那样机械化, 因而把巧而难的证明, 化为计算那样虽繁而易的劳动呢? 事实上, 这一证明机械化的设想, 并不始自今日, 它早就为 17 世纪时的大哲学家、大思想家和大数学家 Descates 和 Leibniz 所具有. 只是直到 19 世纪末, Hilbert(德国数学家, 1862~1943)等创立并发展了数理逻辑以来, 这一设想才有了明确的数学形式. 又由于 20 世纪 40 年代电子计算机的出现, 才使这一设想的实现有了现实可能性.

从 20 世纪二三十年代以来, 数理逻辑学家们对于定理证明机械化的可能性进行了大量的理论探讨, 他们的结果大都是否定的. 例如 Gödel 等的一条著名定理就说, 即使看来最简单的初等数论这一范围, 它的定理证明的机械化也是不可能的. 另一面, 1950 年波兰数学家 Tarski 则证明了初等几何(以及初等代数)这一范围的定理证明, 却是可以机械化的. 只是 Tarski 的结果近于例外, 在初等几何及初等代数以外的大量结果都是反面的, 即机械化是不可能的. 1956 年以来美国开始了利用电子计算机做证明定理的尝试. 1959 年王浩先生设计了一个机械化方法, 用计算机证明了 Russell 等著的《数学原理》这一经典著作中的几百条定理, 只用了 9 分钟, 在数学与数理逻辑学界引起了轰动. 一时间, 机器证明的前景似乎非

常乐观. 例如 1958 年时就有人曾经预测：在 10 年之内计算机将发现并证明一个重要的数学新定理. 还有人认为，如果这样，则不仅许多著名哲学家与数学家如 Peano、Whithead、Russell、Hilbert 以及 Turing 等人的梦想得以实现，而且计算将成为科学的皇后，人类的主人!

然而，事情的发展却并不如预期那样美好. 尽管在 1976 年，美国的 Hanker 等人，在高速计算机上用了 1200 小时的计算时间，解决了数学家们 100 多年来所未能解决的一个著名难题——四色问题，因此而轰动一时，但是，这只能说明计算机作为定理证明的辅助工具有着巨大潜力，还不能认为这样的证明就是一种真正的机器证明. 用王浩先生的说法，Hanker 等关于四色定理的证明是一种使用计算机的特例机证，它只适用于四色这一特殊的定理，这与所谓基础机器证明之能适用于一类定理者有别. 后者才真正体现了机械化定理证明，进而实现机器证明的实质. 另一面，在真正的机械化证明方面，虽然 Tarski 在理论上早已证明了初等几何的定理证明是能机械化的，还提出了据以造判定机也即是证明机的设想，但实际上他的机械化方法非常繁，繁到不可收拾，因而远远不是切实可行的. 1976 年时，美国做了许多在计算机上证明定理的实验，在 Tarski 的初等几何范围内，用计算机所能证明的只是一些近于同义反复的“儿戏式”的“定理”. 因此，有些专家曾经发出过这样悲观的论调：如果专依靠机器，则再过 100 年也未必能证明出多少有意义的新定理来.

### 3. 一条切实可行的道路

1976 年冬，我们开始了定理证明机械化的研究. 1977 年春取得了初步成果，证明初等几何主要一类定理的证明可以机械化. 在理论上说来，我们的结果已包括在 Tarski 的定理之中. 但与 Tarski 的结果不同，我们的机械化方法是切实可行的，即使用手算，依据机械化的方法逐步进行，虽然繁复，也可以证明一些艰深的定理.

我们的方法主要分两步，第一步是引进坐标，然后把需证定理中的假设与终结部分都用坐标间的代数关系来表示. 我们所考虑的定理局限于这些代数关系都是多项式等式关系的范围，例如平行、垂直、相交、距离等关系都是如此. 这一步可以叫做几何的代数化. 第二步是通过代表假设的多项式关系把终结多项式中的坐标逐个消去，如果消去的结果为零，即表明定理正确，否则再作进一步检查. 这一步完全是代数的，即用多项式的消元法来验证.

上述两步都可以机械与刻板地进行. 根据我们的机械化方法编成程序，以在计算机上实现机器证明，并无实质上的困难. 事实上数学所某些同志以及国外的王浩先生都曾在计算机上试行过. 我们自己也曾在国产的长城 203 台式机上证明了像 Simson 线那样不算简单的定理. 1978 年初我们又证明了初等微分几何中主要的一类定理证明也可以机械化. 而且这种机械化方法也是切实可行的，并据此用

手算证明了不算简单的一些定理.

从我们的工作中可以看出, 定理的机械化证明, 往往极度繁复, 与通常既简且妙的证明形成对照, 这种以量的复杂来换取质的困难, 正是利用计算机所需要的.

在电子计算机如此发展的今天, 把我们的机械化方法在计算机上实现不仅不难, 而且有一台微型的台式机也就够了. 就像我们曾经使用过的长城 203, 它的存数最多只能到 234 个 10 进位的 12 位数, 就已能用以证明 Simon 线那样的定理. 随着超大规模集成电路与其他技术的出现与改进, 微型机将愈来愈小型化而内存却愈来愈大, 功能愈来愈多, 自动化的程度也愈来愈高. 进入 21 世纪以后, 这一类方便的小型机器将为广大群众普遍使用. 它们不仅将成为证明一些不很简单的定理的武器, 而且还可用以发现并证明一些艰深的定理, 而这种定理的发现与证明, 在数学研究手工业式的过去, 将是不可想象的. 这里我们应该着重指出, 我们并不鼓励以后人们将使用计算机来证明甚至发现一些有趣的几何定理. 恰恰相反, 我们希望人们不再从事这种虽然有趣却即是对数学甚至几何学本身也已意义不大的工作, 而把自己从这种工作中解放出来, 把自己的聪明才智与创造能力贯注到更有意义的脑力劳动上去.

还应该指出, 目前我们所能证明的定理, 局限于已经发现的机械化方法的范围, 例如初等几何与初等微分几何之内. 而如何超出与扩大这些机械化的范围, 则是今后需要探索的长期的理论性工作.

### 4. 历史的启示与中国古代数学

我们发现几何定理证明的机械化方法是在 1976 至 1977 年之间. 约在两年之后我们发现早在 1899 年出版的 Hilbert 的经典名著《几何基础》中, 就有着一条真正的正面的机械化定理: 初等几何中只涉及从属与平行关系的定理证明可以机械化. 当然, 原来的叙述并不是以机械化的语言来表达的, 也许就连 Hilbert 本人也并没有对这一定理的机械化意义有明确的认识, 自然更不见得有其他人提到过这一定理的机械化内容. Hilbert 是以公理化的典范而著称于世的, 但我认为, 该书更重要处, 是在于提供了一条从公理化出发, 通过代数化以到达机械化的道路. 自然, 处于 Hilbert 以及其后数学的一张纸一支笔的手工作业时代里, 公理化的思想与方法得到足够的重视与充分的发展, 而机械化的方向与意义受到数学家的忽视是完全可以理解的. 但电子计算机已日益普及, 因而繁琐而重复的计算已成为不足道的事情, 机械化的思想应比公理化思想受到更大重视, 似乎是合乎实际的.

其次应该着重指出, 我们从事机械化定理证明工作获得成果之前, 对 Tarski 的已有工作并无接触, 更没有想到 Hilbert 的《几何基础》会与机械化有任何关系. 我们是在中国古代数学的启发之下提出问题并想出解决办法来的.

说起来道理也很简单：中国的古代数学基本上是一种机械化的数学. 四则运算与开方的机械化算法由来已久. 汉初完成的《九章算术》中, 对开平、立方与解线性联立方程组的机械化过程, 都有详细说明. 宋代更发展到高次代数方程求数值解的机械化算法.

总之, 各个数学领域都有定理证明的问题, 并不限于初等几何或微分几何. 这种定理证明肇始于古希腊的 Euclid 传统, 现已成为近代纯粹数学或核心数学的主流. 与之相异, 中国的古代学者重视的是各种问题特别是来自实际要求的具体问题的解决. 各种问题的已知数据与要求的数据之间, 很自然地往往以多项式方程的形式出现. 因之, 多项式方程的求解问题, 也就自然成为中国古代数学家研究的中心问题. 从秦汉以来, 所研究的方程由简到繁, 不断有所前进, 有所创新. 到宋元时期, 更出现了一个思想与方法的飞跃：天元术的创立.

“天元术”到元代朱世杰时又发展成四元术, 所引入的天元、地元、人元、物元实际上相当于近代的未知元或未知数. 将这些未知元作为通常的已知数那样加减乘除, 就可得到与近代多项式与有理函数相当的概念与相应的表达形式与运算法则. 一些几何性质与关系很容易转化成这种多项式或有理函数的形式及其关系. 这使得过去依题意列方程这种无法可循需要高度技巧的工作从此变成轻而易举. 朱世杰 1303 年的《四元玉鉴》又给出了解任意多至四个未知元的多项式方程组的方法. 这里限于 4 个未知元只是由于所使用的计算工具(算筹和算板)的限制. 实质上他解方程的思想路线与方法完全可以适用于任意多的未知元.

不问可知, 在当时的具体条件下, 朱世杰的方法有许多缺陷. 首先, 当时还没有复数的概念, 因之朱世杰往往限于求出(正)实值. 这无可厚非, 甚至在 17 世纪 Descartes 的时代也还往往如此. 但此外朱世杰在方法上也未臻完善. 尽管如此, 朱世杰的思想路线与方法步骤是完全正确的, 我们在 20 世纪 70 年代之末, 遵循朱世杰的思想路线与方法的基本实质, 采用美国数学家 J.F.Ritt 在 1932, 1950 年关于微分方程代数研究书中所提供的某些技术, 得出了解任意复多项式方程组的一般算法, 并给出了全部复数解的具体表达形式. 此后又得出了实系数时求实解的方法, 为重要的优化问题提供了一个具体的方法.

由于多种问题往往自然导致多项式方程组的求解, 因而我们解方程的一般方法可被应用于形形式式的问题. 这些问题可以来自数学自身, 也可以来自其他自然科学或工程技术. 在本丛书的第一本书, 吴文俊的《数学机械化》一书中, 可以看到这些应用的实例. 在工程技术方面的应用, 在本丛书中已有高小山的《几何自动作图与智能 CAD》与陈发来和冯玉瑜的《代数曲面拼接》两本专著. 上述解多项式方程组的一般方法已推广至代微分方程的情形. 许多应用以及相应论著正在酝酿之中.

**5. 未来的技术革命与时代的使命**

宋元时代天元术与四元术的创造, 把许多问题特别是几何问题转化成代数方程与方程组的求解问题. 这一方法用于几何可称为几何的代数化. 12 世纪的刘益将新法与“古法”比较, 称“省功数倍”, 这可以说是减轻脑力劳动使数学走上机械化的道路的一项伟大的成就.

与天元术的创造相伴, 宋元时代的数学又引进了相当于现代多项式的概念, 建立了多项式的运算法则和消元法的有关代数工具, 使几何代数化的方法得到了系统的发展, 见于宋元时代幸以保存至今的杨辉、李冶、朱世杰的许多著作之中. 几何的代数化是解析几何的前身, 这些创造使我国古代数学达到了又一个高峰. 可以说, 当时我国已到达了解析几何与微积分的大门, 具备了创立这些数学关键领域的条件, 但是各种原因使我们数学的雄伟步伐就在这些大门之前停顿下来. 几百年的停顿, 使我们这个古代的数学大国在近代变成了数学上的纯粹入超国家. 然而, 我国古代机械化与代数化的光辉思想和伟大成就是无法磨灭的. 本人关于数学机械化的研究工作, 就是在这些思想与成就启发之下的产物, 它是我国自《九章算术》以迄宋元时期数学的直接继承.

恩格斯曾经指出, 枪炮的出现消除了体力上的差别, 使中世纪的骑士阶级从此销声匿迹, 为欧洲从封建时代进入到资本主义时代准备了条件. 近年有些计算机科学家指出, 个人用计算机的出现, 其冲击作用可与枪炮的出现相比. 枪炮使人们在体力上难分强弱, 而个人用计算机将使人们在智力上难分聪明愚鲁. 又有人对数学的未来提出看法, 认为计算机的出现, 将使数学现在一张纸一支笔的方法, 在历史的长河中, 无异于石器时代的手工方法. 今天的数学家们, 不得不面对计算机的挑战, 但是, 也不必妄自菲薄. 大量繁复的事情交给计算机去做了, 人脑将仍然从事富有创造性的劳动.

我国在体力劳动的机械化革命中曾经掉队, 以致造成现在的落后状态. 在当前新的一场脑力劳动的机械化革命中, 我们不能重蹈覆辙. 数学是一种典型的脑力劳动, 它的机械化有着许多其他类型脑力劳动所不及的有利条件. 它的发扬与实现对我国的数学家是一种时代的使命. 我国古代数学的光辉, 鼓舞着我们为实现数学的机械化, 在某种意义上也可以说是真正的现代化而勇往直前.

吴文俊

2002 年 6 月于北京

# 序　言

由于计算机硬件和软件等并发系统的规模不断扩大，并发系统的组成和结构越来越复杂，高效正确地设计、分析和开发这样的系统变得越来越困难，因此，必须对并发系统的结构进行简化处理. 然而，复杂并发系统的简化处理存在许多理论和技术上的困难，同样，在系统设计和开发方面也存在着大量的理论和技术障碍.加强这方面的理论和技术研究，无论对计算机科学技术的发展，还是对计算机硬件和软件的生产都具有理论和现实意义.

形式化方法是在 20 世纪 60 年代末为了解决“软件危机”而发展起来的. 所谓的形式化方法，就是利用具有严格数学基础的刻画语言和分析技术来设计、分析和简化计算机系统. 形式化方法可以简单地理解为形式刻画和形式推理相结合的一种研究方法. 形式刻画就是要明确描述所要设计的系统及其性质，通常系统的刻画由一种形式语言，例如进程代数来实现，这种刻画语言具有严格的语法和语义，用来表达系统的功能行为或者内部结构等. 由于形式刻画具有严格的数学基础，因此，消除模糊性、二义性和不完整性利于系统的设计、分析推理和简化处理. 形式化方法的重要应用就是设计、分析和简化并发系统，高效机械化的系统形式化设计、分析与简化方法是它所追求的目标.

作为控制并发系统复杂度和保证并发系统设计正确性和服务质量的一种有效手段，形式化方法的研究经过几十年的努力，取得了很大成绩，相应技术已经应用到了计算机软件和硬件系统设计的各个方面. 著名的例子如 IBM 的 CICS 系统、AT&T 的交换机系统、欧洲铁路跟踪系统、空中客车 A330/340 机舱通信系统等.

计算机硬件和软件等并发系统中存在大量的相同或同构的组件，相应地，它们的模型常常在结构上展示出对称性，可以在任何带有重复的系统，例如存储器、高速缓存、总线协议、网络协议中找到对称结构. 这种对称结构 一般来说具有相同或非常相近的性质. 基于这一重要特性，人们希望发展对称约简方法以达到简化模型的目的，特别是在并发系统的形式化验证技术当中，对称约简已经成为解决状态爆炸问题的有效方法之一. 进程代数是一种刻画并发系统的最常用的数学描述语言，事件结构可以作为能够刻画真并发的进程代数模型，它们的结构常常展示出对称性. 然而，当前大多数的研究都集中在基于变迁系统模型的对称约简，从未涉及进程代数及其事件结构模型的结构对称性.

本书对进程代数语言及其事件结构模型中的对称性和对称约简方法进行了深

入探讨, 力图从结构上建立对建模语言和模型进行约简的基本理论和方法, 目的是希望从结构上建立起类似于行为等价的由细到粗的约简体系, 为实现高效机械化并发系统设计和分析而需要建立的不同层次静态约简模型服务. 另一方面, 自顶向下逐步细化的层次化方法是人们广泛接受的计算机硬件和软件并发系统机械化设计的最主要的方法之一. 动作细化是并发系统层次化刻画和设计方法的核心操作, 其研究可以在进程代数和事件结构模型中进行, 已经取得了丰硕的成果. 在此基础上, 本书讨论了对称约简对动作细化的影响, 交织等价和步进等价在动作细化下的保持问题, 以及基于一种简单对称性思想的偏序约简方法及其在模型检验中的应用. 具体如下：

- 针对进程代数语言, 定义了进程的对称性概念, 建立了对称约简算法, 并证明了约简后的进程与原进程交织迹和交织互模拟等价, 同时提供了有意义的实例说明对称性的定义, 并验证了约简算法的正确性.

- 针对事件结构模型, 提出了基于置换群的对称性概念. 在事件结构中, 引入了事件结构的商结构模型, 证明了商结构与原事件结构是迹、互模拟和偏序多集迹等价的, 建立了针对事件结构的对称约简算法, 证明了对称约简不影响等价在动作细化下的保持. 在研究了事件结构中的对称性之后, 进一步从理论上比较了对称约简与自互模拟约简的区别和联系.

- 交织等价(即交织迹等价和交织互模拟等价)与步进等价(即步进迹等价和步进互模拟等价)在动作细化下是不保持的. 一些研究工作证明了交织互模拟等价在严格限制动作细化的情况下才能保持, 但在这种限制下交织迹等价仍然不保持, 而且没有工作进一步讨论步进迹等价和步进互模拟等价在动作细化下的保持问题. 发现了一类具有特定性质的并发系统，它们能使交织等价和步进等价在没有限制动作细化的条件下保持.

- 提出了“束动作变迁”的概念, 这种处理进程变迁的方式不同于基于迹理论采用偏序约简以减少并发模型状态数目的方法, 而是将完全并发的动作看成一个“大动作”, 使得进程在该大动作执行的前后仅存在两个状态, 因而束动作变迁的概念可用于处理并发系统验证过程中出现的状态爆炸问题. 将束动作变迁的概念应用到了基于偏序约简的模型检验中, 讨论了改进后的偏序约简方法在模型检验中的应用.

这些成果的获得为并发系统的设计、分析和简化处理提供了有利武器.

**致　谢**

衷心感谢中国科学院数学与系统科学研究院的吴文俊院士, 是他将我带入了数学与计算机科学的交叉领域, 他关于数学与计算机科学的深邃思想与见解使我对理论计算机科学产生了浓厚的兴趣. 感谢已故的北京大学的程民德院士, 在他的指导下, 我对信息与计算机科学有了更加深刻的认识. 感谢已经故去的德国 Max-Planck 计算机科学研究所的 Harald Ganzinger 教授, 我在 Max-Planck 度过了两年的美好时光, 每天中午午饭后的咖啡屋聚会使作者在形式逻辑学方面受益匪浅.

德国 Mannheim 大学的 Mila Majster-Cederbaum 教授与我长达六年在形式化方法领域的合作研究是本书的基础, 在此表示衷心感谢. 同时感谢德国 Mannheim 大学的 Frank Sagler 博士以及 Kiel 大学的 Harald Fecher 博士, 本书得益于与他们的关于动作细化的大量讨论与合作研究.

长期以来支持和帮助过我们的有中国科学院数学与系统科学研究院的高小山研究员、刘卓军研究员、石赫研究员、中国科学院软件研究所的林惠民院士、成都计算机应用研究所的张景中院士、杨路研究员以及美国 Texas A&M 大学的 Mi Lu 教授，在此一并致谢.

对审阅本书以及在本书写作和出版过程中给予热情帮助的科学出版社编辑表示感谢. 感谢提供支持和帮助的所有同事、家人和朋友.

吴尽昭

2006 年 6 月

# 目　　录

# 第一章　绪　　论

Whenever you have to do with a structure-endowed entity, try to determine its group of automorphisms.

——Hermann Weyl[99]

随着计算机软硬件等并发系统规模的不断扩大，其组成和结构越来越复杂，设计、分析和开发并发系统变得越来越困难. 因此必须对系统的语言刻画和模型进行合适的简化处理，才能保证不断发展的系统设计与分析的需求.

## 1.1　对　　称

“对称”一词译自英文单词“symmetry”，它的原意是“均匀”和“完美”. 这个词被人们不断推移和引申，其应用范围早已超出了原意. 科学和艺术都很重视对称性. 对于科学，对称性决定了各种可能的守恒定律，因而具有更根本性的意义. 在艺术中，对称性常与平衡、形状、形式、空间等一同讨论. 人们通常从静态表现上理解对称性，有一定意义，但更重要的是从操作意义上、从生成过程上理解对称性.

在科学中，对称性是指某种操作下的不变性或者守恒性，对称性常与守恒定律相联系. 与空间平移不变性对应的是动量守恒定律；与时间平移不变性对应的是能量守恒定律；与转动变换不变性对应的是角动量守恒；与空间反射 (镜像) 操作不变性对应的是宇称守恒.

艺术作品有意识或无意识地利用对称，其复杂性不亚于目前科学上的对称性，有许多还不能作科学分析. 以音乐和文学为例，它们中存在许多重复的词、句子和段落. 重复就是一种简单的对称. 但如何重复，在何种水平或层次上重复，则有很多变化.

无论对于科学还是对于艺术，对称性都涉及不同的方面和不同的层次. 不同方面指对称的多样性：平移对称 (连续装饰花纹、花布)、旋转对称 (穹窿、五角星、伞、晶体)、左右对称 (建筑立面、人体) 及联合操作对称等. 不同方面还涉及局部与整体的关系，对称性有长程整体对称 (如晶体)，也有局部短程对称 (如准晶)，这些在科学与艺术作品中都有许多实例. 不同层次指对称性依赖于物质层次或者观念层次，在不同的层次上对称性可以很不相同，以人体为例，外表是左右对称的，但内脏则不是，心脏通常靠近左侧，肾等则是对称的.

从动态的观点看对称性，我们获得了运动、变化与发展的印象，最终是“生成”的观念. 对称性对于自然界是一种自组织机制，一方面或层次的对称性的破缺可能

为另一方面或层次的对称性的存在提供条件或者基础，反之亦然. 不仅无机界构成及演化受对称性支配，有机界、生命界也是如此.

### 1.1.1 计算机科学中的对称

对称问题广泛存在于科学的各个分支，研究对称问题的目的，是想通过对被研究对象的部分性质的研究，来推断全体的性质. 对称概念中最原始、最直观的是几何图形对称概念.

计算机硬件和软件系统中都存在大量的相同或同构的组件，比如在内存中存在相同的单元，在程序模块中存在相同的语句，在总线协议和网络协议中存在重复相同的结构. 如果相同或同构的组件存在，在当前系统中存在相同的功能，甚至相同的性能，就称当前系统存在对称结构，具有对称性. 另一方面，刻画计算机系统的语言和模型也存在对称性，比如在进程代数中，选择操作算子左右两边的进程是对称的，完全并行操作算子的左右两边的进程也可以说存在对称关系；再比如，在事件结构模型中，事件的矛盾关系、独立关系都是对称的. 目前的计算机硬件和软件系统结构越来越复杂，更有可能存在大量的相同或同构的组件. 为设计、分析和开发计算机系统的需要，必须用语言和模型来刻画和描述计算机系统，必然会存在一些对称的部分，其中有些对称部分的分析完全是多余的、无用的. 因此，为正确地设计、分析、开发计算机系统，必须首先从结构上对其进行约简，这种约简方法一般称为“对称约简”.

### 1.1.2 对称与对称约简

计算机硬件和软件系统中都存在大量的相同或同构的组件，相应地，它们的模型常常在结构上存在对称性，这种对称结构一般来说具有相同或非常相近的性质. 为了使问题简化，人们根据这种情况开发了针对结构约简的对称约简方法来简化模型. 例如，在计算机系统形式化验证方法中，状态空间探测技术能够检测有限状态系统中的错误. 然而主要难以克服的问题是，状态空间的大小会随着模型的增大而迅速增长，即所谓的状态爆炸问题，对称约简方法已经成为解决状态爆炸问题的有效方法之一.

本书主要讨论计算机系统的刻画语言进程代数及其事件结构模型中的对称性. 具体做法是，用等价关系表示系统中的对称性，这种等价关系建立在置换群上，该置换群中的每一个置换就是一个自同构，它刻画了该系统相同或同构的组件，在此基础上，就称该置换群为自同构群. 本书针对进程代数语言建立了进程的对称性概念，据此建立了对称约简算法，验证了约简算法的正确性，并证明了约简后的进程与原进程是交织迹和交织互模拟等价的. 另一方面，本书采用事件结构对进程代数建模，针对事件结构模型提出了基于置换群的对称性概念. 给定一个事件结构和其上的一个置换群，引入了事件结构的商结构模型，证明了商结构与原事件结构是

迹、互模拟和偏序多集迹等价的，同时建立了针对事件结构的对称约简算法，阐明了对称约简不影响等价在动作细化下的保持. 在讨论了事件结构的对称性之后，进一步从理论上比较了对称约简与自互模拟约简的差别：给定一个事件结构，由对称约简得到的商事件结构与由自互模拟约简得到的互模拟商模型存在许多不同之处. 前者与原事件结构是互模拟等价并偏序多集等价的，而且在动作细化下前者能保留原事件结构的行为. 后者虽然与原事件结构是互模拟等价的，但是不一定是偏序多集迹等价的，同时在动作细化下也不一定能保留原事件结构的行为.

# 1.2 动作细化

众所周知，并发性、分布性已经成为现代复杂计算机软件和硬件系统的基本特征. 人们把这样的由相互作用的子系统构成的，并且明显呈现出并发性、分布性特点的系统统称为并发系统. 现实中的绝大多数计算机系统都是并发系统，如通信系统、大量的基于网络的应用系统、电路系统以及各种各样的嵌入式系统、实时系统等. 由于并发系统的并发特性是通过子系统的相互作用以及系统与环境的交互作用来体现的，因此又把并发系统称为反应式系统.

随着计算机系统规模的不断扩大，计算机系统的结构和组成日益复杂，用传统的设计方法已经不能或难以设计出符合要求的系统. 例如反应式系统，它与外界环境进行复杂的信息交换，并且在大量不同的情况下具有难以预计的反应. 又如实时系统，它不但要按要求提供正确的操作，而且对操作的具体时间有严格的要求. 在大型通信系统中，有成千上万的用户在不断地发出请求，大量的进程在并行工作，这些进程之间又在不断地相互作用. 如何使通信没有阻塞和死锁而保持最大的流量等问题都需要研究新的理论和方法. 因此并发系统的形式化模型、机械化的设计和实现理论、方法及实用技术的研究就成为了现代计算机科学研究的重要课题，引起了数学家和计算机科学家的广泛兴趣，并且逐步发展成一个相对独立的领域 —— 并发理论.

## 1.2.1 动作细化的方法

如何描述并发系统的行为，在顺序系统中是个比较容易的问题，但在并发系统中由于存在交互特性而变得非常棘手. 对于并发系统的行为刻画，一种传统的广为接受的方法是基于抽象状态机的状态变迁方法，例如标记变迁系统：假设系统中的一个活动，在某个抽象层次上是原子的，即不能被中断执行的，它可以用状态间一个变迁来表示，活动名作为标签标记在变迁上，一旦定义了这些原子动作后，人们就可以用“模块化”来控制系统描述的复杂性：即系统可以由“小”的标记变迁系统一步步地构造出来.

然而，从软件工程的角度看，这一理论在很多情况下是不适用的. 其根本原因在于抽象层次是固定的，即定义在某个抽象层次上的原子动作集合在整个系统描述中保持不变. 但是，在基于组件的系统开发技术当中，人们常常需要比较概念上属于两个不同抽象层次上的系统，以便确定它们是否在本质上实现了相同的功能. 对于一个给定了不同抽象层次上的动作集合，一种更有效地控制系统复杂性的技术是层次化的系统刻画和设计方法：对于一个复杂的系统，人们首先给出一个简单的抽象的系统描述，在此基础上进行分析，然后逐层进行，将高层次的简单的系统描述细化为低层次的、实际的、复杂的系统实现，每次关注从上一层引入当前层的相关的细节.

动作细化是系统层次化刻画和设计方法的一种核心操作，是在系统的基本构成单位是动作的思想框架下，实现并发系统层次化的一种基本操作，其基本思想是将系统高层次的基本单位替换为低层次的进程 [1～3,28,29,33,41,62～66,81,85,88,89,96]，即首先用一组简洁、紧凑的动作集合来抽象描述一个简单的并发系统的行为，然后在该层次上选定一个或几个抽象的未加解释的动作，并分别用低一层次的具体的活动 (动作集合) 进行替换，逐层做下去，直至得到详尽的系统设计或实现.

并发系统的动作细化一般在三个层次上进行：结构层次 (例如变迁系统、Petri 网和事件结构 [94])；语言层次 (例如进程代数 CCS[70]、CSP[51]、ACP[5] 和 LOTOS [11])；逻辑层次 (例如 CTL、模态 mu 演算). 无论在哪个层次上实施动作细化，必然涉及三个基本要素：(1) 实现层次化的基本操作；(2) 并发系统的形式化模型；(3) 动作的基本特性. 因此，从不同的视角出发，演变出各种各样的动作细化方法. 下面简单地讨论一下几个重要的概念范畴.

一种实现并发系统层次化的经典方法是将动作细化视为算子的方法，即将细化定义成一个在语法或语义域里类似于替换的显式算子. 这里，将“动作”理解为选定抽象层次上概念实体的一个抽象. 通过把给定抽象层次上的整个系统描述作为动作细化算子的变量，将这一层次的动作解释为低层次的具体的复杂的进程，进而获得低层次的较详尽的整个系统的描述，就可以实现层次化系统描述. 通过这一模式，细化后系统的行为可以由原系统的行为和用来替换动作的系统的行为组合化得到. 需要指出的是，在这一方法框架下，人们把动作当作未加解释的活动. 这样，原始系统的行为 (细化前) 并不包含那些要被细化的动作的任何固有信息. 因此，从这个意义上讲，并不存在一个细化是否“正确”的概念. 文献中大量的工作都是遵循这一方法，如定义在进程代数上的动作细化算子 [1,3,29,49,62,63,81,96] 和定义在语义模型上的动作细化算子 [28,33,41,64～66,88,89].

另一类实现层次化的方法是通过把细化看成是进程之间的关系，而不是从单位动作到进程的算子来定义一个更为松散的细化概念 [47,49,81]. 这类方法正好沿用了由 N.Wirth[95] 最先提出，并在顺序程序设计中得到成功应用的“逐步求精”的思想. 然而，与将动作细化视为算子的观点相比，这类方法有明显的不足：一方面，在

系统刻画中没有清晰地分离不同的抽象层次，没有明确区分哪些动作属于哪一固定的层次；另一方面，对于给定的系统描述，可以有不同的系统实现，因此必须建立给定进程的所有可能细化的实现规则.

### 1.2.2 动作细化的分类

1. 原子与非原子动作细化

动作细化的基本方法可以分为两大类：一种是原子细化 [6,12,29,46,47,59]，这种方法的观点是动作被视为是原子的，不能被中断执行，并且动作细化在某种意义上应该保持这种原子性；另外一种方法是非原子细化，这是一种比较自由的细化概念，原子性总是相对于当前的抽象层次. 由于细化改变设计的层次，所以可能破坏动作的原子性.

为了更好地解释这个问题，看一个简单的例子：$a|||b$，表示动作 $a$ 和 $b$ 的并行操作. $a$ 被细化成 $a_1;a_2$. 图 1.1 是当细化是原子的和非原子的时，$a|||b$ 和 $(a|||b)[a \rightarrow a_1;a_2]$ 的标记变迁系统. 在这个图中，黑点表示抽象的可观察状态 (例如，观察到的相关状态，其中每个原子动作都已完成)，空圈表示具体的内部状态 (例如，在较低抽象层上中间执行步骤的状态，这些状态不能够被外部观察到). 区别是很明显的：如果细化是原子的，那么在 $a_1$ 和 $a_2$ 的执行之间没有可观察到的动作，并且 $b$ 不能够在它们之间被执行.

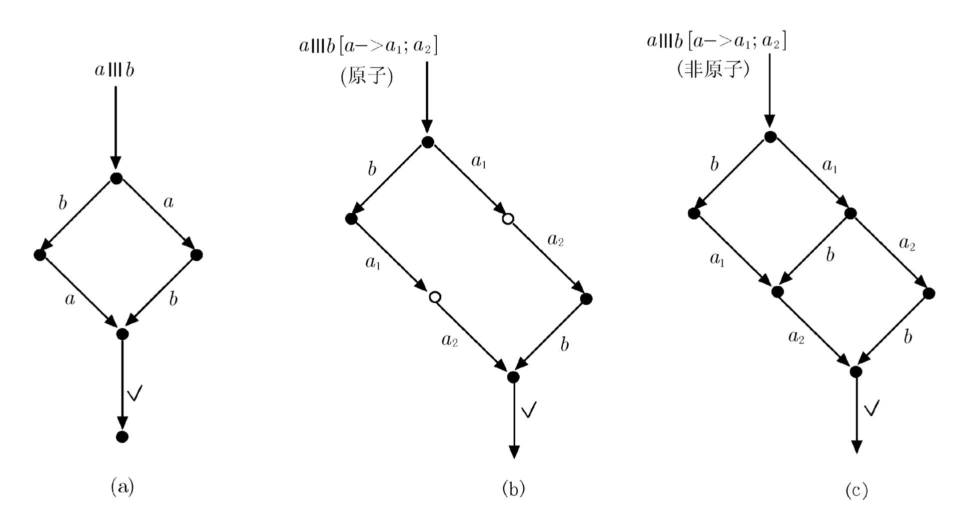



图 1.1 $a|||b$ 的原子与非原子细化

原子的方法看起来非常适合一些实际问题. 例如，当一种语言实现为另外一种语言时，就需要前者的原语言作为后者的程序复合词. 在这种情况下，执行过程中保持原子性是执行正确性的重要保障因素. 这种例子在文献 [48] 中可以找到，其中

Milner 的 CCS 被匹配成一种颗粒度细小的演算，每个 CCS 的转移，被作为后者适合的事务 (原子执行的变迁序列) 来实现. 另外一种支持原子细化的例子是，必须用资源互斥来阻止错误的行为. 例如，对共享变量 $x$ 的一个抽象写操作 $a$，当动作 $a$ 细化为复杂的进程 $u$ 时，只有在 $u$ 完成之后，才能对 $x$ 进行读或者写操作. 关于原子细化更多的实例可以在文献 [12] 中找到.

另一方面，非原子细化方法也有大量的实际应用 [3,17,31,37,52,57,72,82,88]. 实际上，这种方法比原子方法在总体上更为实用. 例如，图 1.1 中，如果动作 $b$ 是一个与 $a$ 完全独立的动作，那么当执行 $a_1a_2$ 序列时，让 $b$ 处在空闲等待状态，加上这种限制显然是不合理的. 原子和非原子之间的选择应该根据实际应用来决定.

2. *语法和语义动作细化*

对动作细化基本上有两种解释：语法和语义. 在语法细化方法中，动作细化非常类似于顺序程序中过程调用的复制规则. 例如，文献 [72] 使用静态复制规则 (执行以前语法替换)；而文献 [2] 采用动态复制规则：当执行 $t$ 时，一旦 $a$ 发生，那么 $u$ 的第一个动作被执行，到达状态 $u'$，那么在这之后，$t$ 中的动作 $a$ 被 $u'$ 语法替换. 在以上任何一种情况下，$t[a \to u]$ 的语义是项 $t\{u/a\}$ 的语义. 在其他情况下，正如在 ACP 中的研究 [5]，它的含义是进程代数具有一个顺序组合操作 (而不是比较标准的动作前缀操作算子). 因为在必然的语法替换下，它是不封闭的. 在语义解释中，替换操作被定义在语义域中，用来解释项. 那么 $t[a \to u]$ 的语义可以使用域中的符号来定义. 例如，当使用事件结构作为语义域时，在事件结构 $[\![t]\!]$ 中，事件结构 $\mathcal{E} = [\![u]\!]$ 表示 $u$ 的语义，对每一个标识为 $a$ 的事件用 $d$ 将其替换. 细化操作保持事件嵌入语义：例如，如果事件 $d$ 与事件 $e$ 相矛盾，那么 $\mathcal{E}$ 中的所有事件也与 $e$ 相矛盾，同样顺序关系也相似. 这种细化操作算子的研究可以在文献 [10,17,18,25,26,31,35,37,39,40,56,82,88,89] 中找到，这些文献选择的语义模型有基本 (prime)，自由 (free)，流 (flow) 和稳定 (stable) 事件结构，格局 (configuration) 结构，偏序集合集，同步树，原因树，ST 树和 petri 网. 支持语义替换方法的研究认为开始点是引入语义细化的概念作为在语义模型中的纯替换. 通常，这点不是非常困难的. 相反，困难的地方是对语法细化操作算子寻找操作定义，这种语法细化正确地执行语义替换. 例如，顺序实例中的复制规则的并发对应部分.

上述两种方法是根本不同的. 本质上，在 $t$ 中动作 $a$ 被 $u$ 语法替换，会产生抽象层次的混乱，然而语义替换不会产生这种情况. 这种混乱可能来自于进程之间产生的新通讯，而在较高层次，这种混乱不会产生. 概念上，通常这是不需要的. 这种情况阻止了对动作细化的代数理论定义. 技术上，语法细化对应于代数之间的同态，代数的符号差由语言来定义 (因为同态产生于从一个动作到子项的匹配，子项需要被分布在所有的操作算子上)；另一方面，语义细化定义了在语义模型上的一个操作，这个语义模型是可组合的. 因为两种方法不一致，我们不能期望定义是可

组合的同态. 文献 [90] 比较了这两种方法，并确定在什么限制下，二者可以产生相同的结果. 这个结果不仅能让我们清楚地理解动作细化理论，而且也让动作细化应用者知道什么时候语义细化可以在概念上通过简单的语法替换来实现.

3. 交织和真并发语义动作细化

当动作细化是一种语言的操作算子时，很自然地，一个关键问题就是要找到一种与这样的操作算子同余的等价概念. 形式上，给定一个候选的等价概念 $\simeq$，我们想找到包含在 $\simeq$ 中的最粗的关系 $\equiv$，它与这种语言的所有的操作算子都是同余的. 精确地说，就是：

- 无论什么时候 $u_1 \equiv u_2$，那么 $t[a \to u_1] \equiv t[a \to u_2]$；

- 无论什么时候 $t_1 \equiv t_2$，那么 $t_1[a \to u] \equiv t_2[a \to u]$.

同余问题的前一半看起来比较容易，主要的要求就是要清楚地区分死锁 (在这种状态下，系统什么也不做) 和终止 (在这种状态下，系统除了终止外也什么都不做，如放弃控制)[38]；如果把终止作为一种特殊的动作 (本书中记为 $\surd$)，这种区分很容易做到. 同余问题不得不做这种区分的原因可以通过一个例子来加以理解. 记 $t = a;b$, $u_1 = c$ 表示 $c$ 的执行导致成功的终止，$u_2 = c;\ \ 0$ 表示 $c$ 的执行引起了死锁. 当不考虑结束时，$u_1$ 和 $u_2$ 是等价的；但是，$t[a \to u_1]$ 不能执行 $b$，$t[a \to u_2]$ 却可以. 对于同余问题的后一半，解决方法可以很简单 (当处在交叉语义之内时)，也可以很困难 (当被迫转移到真并发语义上时)，这取决于施加在这种操作算子之上的假设和代数性质.

- 可以在变迁系统上把语义动作细化定义为一种操作算子，这对强互模拟是明确的. 无论如何，应该清楚的是并没有本质的原因使得交织关系不能和动作细化同余.

- 然而，上面提及的操作算子不能满足一个很直观且很重要的特性：它不能在并行组合操作上进行分布. 如果想让这种特性得以保持，最简单的方法就是用原子细化来代替它 [29]. 强和弱的互模拟对这种算子是同余关系. 代价是要么去区分具体的和抽象的状态，要么是用动作序列而不是单一动作作为变迁标记.

- 原子细化并不总是正确的. 如果要求非原子细化算子能分布在并行组合上，这也可定义在标准的变迁系统上，但不允许对决定选择的动作和与它自己并发执行的动作进行细化 [24]. 再者，强互模拟对这个最终算子是同余关系. 由于对可细化动作的限制，这种算子不再分布在选择操作上. 在后面将进一步探讨这个问题.

- 如果想让动作细化分布在并行组合上，并且是非原子的，可应用到所有动作，而不管它在运行中的位置，那么很有必要使用一个比标准变迁系统更有表现力的模型. 文献 [18,74] 最早谈及这个问题.

考虑 $a|||b$ 和 $a;b+b;a$，它们在交织语义中是等价的：前者表示动作 $a$ 和 $b$ 的并发执行，后者表示它们以任意次序执行. 当我们将 $a$ 细化成 $a_1;a_2$，随之相应的进程分别等价为 $(a_1;a_2)|||b$ 和 $a_1;a_2;b+b;a_1;a_2$；由于只有前者能执行 $a_1ba_2$，所以这些动作在交叉语义中就不等价了. 这意味着所需要的同余 $\equiv$ 不能使得 $a|||b$ 和 $a;b+b;a$ 等价.

这个问题在关于动作细化的文献中受到了大量的关注. 一种解决方案是转移到所谓的真并发语义上，如带有比标准交织语义更多关于系统活动并发信息的语义模型. 例如，基于事件的模型 [93,94] 就被用来处理这个问题 [26,31,37,40,82]. 事件模型的同构能产生同余关系，但是，令人质疑的是，这种等价关系相对于交织关系来说太强了，使得它的区分比需要的要多. 这可以通过将基于事件的模型解释成比同构要弱一点的关系来弥补，比如保持历史互模拟 (history-preserving bisimulation)[30,35,80]，或者考虑弱区分模型，如原因树 [26,27]，结果便是必须增加的最小量的信息，就是区分所有动作的开始和结束，这叫做 ST 原则. 系统详细的综述可以参考文献 [40].

正如 Meyer[68,69] 和 Vogler[91] 指出的那样，寻找动作细化同余性的问题也和另外不同的领域有关系，即进程变量的存在完备性. 需要指明的是为动作细化找一个完整的抽象模型也应是避免的.

### 1.2.3　动作细化的保持

在对并发系统的形式化建模中，希望有支持层次结构的形式化方法，在这种自上而下的开发过程中，系统将从抽象的刻画逐步进化成更具体的实现. 动作细化是系统层次化刻画方法的核心操作，这种技术以分层的方式来描述系统，也就是说，用较低抽象层次上复杂的进程来解释较高抽象层次上的动作，从而改变其抽象层次，最终达到实现层次. 它是用一种更复杂的子系统来替代抽象系统中的动作，人们在进程代数和并发系统的语义模型中已经对这种方法进行了研究，我们在前面已经进行了较详细的介绍.

然而，对于并发系统的开发，首先必须对该系统进行建模，即用高层的描述语言或模型来刻画系统，最后根据这些描述要求实现它. 这一过程常常需要等价来确定系统实现的正确性. 例如，$P$ 是系统的描述 (或模型)，$Q$ 是系统的实现，如果 $P$ 和 $Q$ 等价，即 $P\sim Q$，表明这个实现 $Q$ 是正确的. 在开发过程中，对系统的描述 $P$ 可能由高层到低层、由粗到细被具体化，相应地，实现也可能由框架演化成具体代码或元器件，这些演化就是细化过程，其中的核心操作就是动作细化，实现和描述在各个层面上都需要保持等价，这样才能保证实现的正确性. 这就涉及一个重要的

问题：哪一些等价在动作细化的情况下是保持的?

比如对某一类等价 $\sim$, 给定两个并发系统 $P$ 和 $Q$, 如果 $P \sim Q$, 并且 ref 是一个动作细化函数, 那么下面的情况

$$\mathrm{ref}(P) \sim \mathrm{ref}(Q)$$

是否成立？这就是等价在动作细化下的保持问题. 也就是说，给定一类等价概念，如果两个系统在这类等价概念下是等价的，人们希望这两个系统在动作被细化后，得到的两个系统在这类等价概念下仍然是等价的，就说这类等价在动作细化下是保持的，否则，是不保持的.

在线型和分支时间谱系中，已经知道交织等价 (即交织迹等价和交织互模拟等价) 与步进等价 (即步进迹等价和步进互模拟等价) 在动作细化下是不保持的. 一些研究人员通常研究怎样限制动作细化，使得这些等价在限制的条件下是保持的，并且证明了交织互模拟等价在非常严格的限制条件下才能保持，但在这种限制下交织迹等价仍然不保持. 另外，还没有人进一步讨论步进迹等价和步进互模拟等价的保持问题. 在本书中，我们发现了一类具有特定性质的并发系统，它们能使交织等价和步进等价在没有限制动作细化的条件下是保持的. 也就是说，在没有原因的独立关系的并发系统中，或者在特定并发环境 (系统的所有变迁关系都是束动作变迁) 的情况下，这两类交织等价和两种步进等价被表明在动作细化下是保持的, 其中，我们提出了“束动作变迁”的概念，这种处理系统变迁的方式不同于基于迹理论采用“偏序约简”以减少并发系统的状态，而是将完全并发的动作看成一个“大动作”，使得系统在这个大动作执行的前后仅存在两个状态，因而束动作变迁的概念可用于处理系统验证过程中出现的状态爆炸问题，我们将进一步研究怎样在模型检验方法中实现这种处理动作变迁的方法.

### 1.2.4 对称与动作细化

一般来说，计算机系统中的对称是针对一个计算机系统内部结构提出的一种概念，它属于数学上研究结构对称领域，但作为计算机系统的特殊性，这里的对称主要考虑硬件系统和软件系统都存在大量的相同或同构的组件产生的对称，我们在描述语言进程代数和事件结构模型上建立对称性概念，提出对称约简算法. 这些研究主要是为简化、优化系统服务，最终目的是希望帮助人们更好地开发计算机硬件和软件系统.

在对并发系统的形式化建模中，系统地研究它们的语义等价有利于更好地理解并发系统的重要性质. 对于并发系统的开发，希望有支持层次结构的形式化方法，在这种自上而下的设计中，系统被从抽象的刻画逐步进化成更具体的实现，动作细化是实现并发系统层次化的核心操作，它是用一种更复杂的子系统来替代抽象系统中的动作. 这一过程常常需要等价来确定系统实现的正确性. 然而，用对称约简

来简化、优化系统的设计和实现是否影响到系统最终实现的正确性，以及是否影响到动作细化过程，是我们必须明确的问题. 基于这些考虑，一方面，我们研究了对称约简和一些等价概念的关系，另一方面，我们讨论了对称约简是否影响这些等价在动作细化下的保持问题.

## 1.3 相关工作

关于计算机科学中对称性的研究，有如下工作：Emerson[32] 和 Clarke[21] 等人研究了模型检验技术中的对称约简，他们是以标记变迁系统作为系统模型，不同于我们的进程代数和事件结构模型；Starke[84] 研究了 Petri 网的对称性，但本质上也是基于变迁系统进行处理的；Hermanns 和 Ribaudo[50] 研究了进程代数中的某种对称性，主要处理的是并行运算的组件；Wu 等人 [97,98] 研究了逻辑程序设计中的对称结构，主要目的是拓广逻辑程序的语义.

关于在动作细化下等价的保持问题，相关工作如下：Vogler [88,89] 最先提出处理等价保持问题的基本思想；Czaja,van Glabbeek 和 Goltz[24] 证明了如果在动作细化前的进程不出现选择运算和动作自并发情况，交织互模拟等价在动作细化下是保持的，但是交织迹等价仍然不保持. Goltz 和 Wehrheim[45] 证明了历史保持的交织互模拟与具有全局依赖关系的原因测试关系是一致的，但没有更进一步讨论动作细化下的保持问题，并且没有讨论具有动作独立关系环境下将会出现的其他情况. van Glabbeek 和 Goltz 在文献 [41] 中总结了最近十多年动作细化的研究成果，对动作细化下的等价保持问题作出了较多解释和论述，证明了交织等价在一般的动作细化条件下不保持，但没有更进一步的讨论. 文献 [53] 扩展了这些工作，深入讨论了步进等价在动作细化下的保持问题.

传统的偏序约简算法在模型检验中的应用方面，人们做了大量的工作，其中具有代表性的研究工作有 Valmari[87],Godefroid 与 Wolper[43,44] 和 Peled[75]. 对传统的偏序约简算法的改进，一般从两方面进行，一方面对算法本身进行改进，主要是加上一些启发式规则 [22]，使得模型检验能快速顺利进行；另一方面是与其他技术相结合进行改进，例如与 on-the-fly 模型检验 [75] 结合，或与符号模型检验 [4] 结合. 这些改进都停留在对动作粒度上的改进，没有提出比动作粒度更大的束动作概念.

## 1.4 本书贡献

本书的主要贡献如下：

- 提出了进程代数中进程的对称性概念，建立了对称约简算法；

- 提出了事件结构模型的对称性概念，建立了对称约简算法；
- 阐述了对称约简与自互模拟约简的区别和联系；
- 找到了一类事件结构，在其上建立的交织等价和步进等价在动作细化下是保持的；
- 提出了一种改进偏序约简的方法 —— 基于束动作变迁的偏序约简方法，并探讨了如何把它应用于模型检验.

这些工作力图从结构上建立对建模语言和模型进行约简的基本理论，目的是希望从结构上建立类似于行为等价的由细到粗的约简体系，为开发并发系统而需要建立的不同层次静态约简模型服务. 同时，进一步完善动作细化理论，解决交织等价和步进等价在动作细化下的保持问题，并将其中得到的部分成果用于改进偏序约简方法，取得实际应用的支持，为进一步研究拓展思路.

## 1.5 本书组织

第二章介绍了进程代数、事件结构模型、动作细化、标记变迁系统和模型检验的基本概念和事实，这一章内容主要是为全文提供所需要的基础知识.

第三章提出了进程代数中的对称性概念，建立了对称约简算法，证明了约简后的进程与原进程是交织迹和交织互模拟等价的，最后给出了两个实例说明对称约简的思想.

第四章提出了事件结构模型中的对称性概念，证明了商事件结构与原事件结构交织迹、交织互模拟和偏序多集迹等价，阐明了商事件结构与原事件结构之间的交织迹、交织互模拟和偏序多集迹等价在动作细化下是保持的，建立了对称约简算法，并以实例进行了说明.

第五章基于事件结构模型揭示了对称约简与自互模拟约简的区别和联系，阐明了对称约简是非常严格的约简，但与行为约简也存在一致性.

第六章讨论了交织等价和步进等价在动作细化下的保持问题，一般来说，交织等价和步进等价在动作细化下是不保持的，本书定义了一类事件结构，在这类事件结构上定义的交织等价和步进等价在动作细化下保持.

第七章提出了一种改进偏序约简的方法 —— 基于束动作变迁的偏序约简方法，并讨论了如何把它应用于模型检验方法当中.

# 第二章　理论基础

## 2.1　进程代数

并发理论是指计算机科学中并行和分布式系统理论，进程代数是并发理论中的一个研究领域. 进程代数的研究最早要追溯到 20 世纪 70 年代早期，主要目的是描述时间 [8]. “进程代数 (process algebra)” 一词最先是在 1982 年由 Bergstra 和 Klop[9] 使用的. 进程代数是泛代数中的一种结构，它满足特殊的公理集. 自 1984 年起，进程代数就被视为一个科学研究领域. 进程代数有时候指用 Bergstra 和 Klop 的代数方法研究并发进程 [5]，有时候指用一般的代数方法研究并发进程. 然而，进程代数现在已经有不同的意义. 首先，考虑 “进程 (process)” 这个词，它指的是系统的行为. 系统是展示行为的系统，特别是一个软件系统的执行、一个机器的动作，甚至一个人的动作更是如此. 行为是一个系统执行的动作或事件的总和，这些动作或事件的执行可能是定时的或随机的. 通常，为了研究需要，只考虑 “真实” 行为的一种抽象，而不考虑其他方面，而且，我们常常只注意行为被观察得到的结果，动作则成为被观察的单位，因此动作被视为离散的，即一个动作在某一时刻出现，并且不同的动作在时间上被分离. 所以，一个进程有时被称为一个离散事件系统 (事件就是动作的出现). “代数 (algebra)” 一词指用代数的或公理的方法讨论行为. 所以，可以说，进程代数是用代数的方法研究系统行为的一门学科.

目前，进程代数是一种形式化地描述复杂并发系统的建模工具，是一种高层的描述语言，是支持并发分布系统的组合描述和其性质形式化证明的代数语言. 它以代数形式来描述模型，并为模型化系统定义了一套完整的语法和语义.

进程代数有很多种，其中主要的有 Bergstra 和 Klop 的 ACP(algebra of communicating processes)[5], Hoare 的 CSP(communication sequential processes)[51], Milner 的 CCS(calculus of communicating systems)[70] 和 ISO 的 LOTOS(language of temporal ordering specifications)[11] 等. 这些进程代数可以通过附加时间 (时间进程代数 (timed process algebra)) 或概率信息 (概率进程代数 (probabilistic process algebra)) 等指标加以扩展，使其表达力更强，可用于并发系统性能评价，比如 PEPA(performance evaluation process algebra) 等.

为了简化问题，我们只讨论这些进程代数语言的一个公共子集，命名为 **L** 语言，它是进程代数语言**最基本**的部分，仅仅包含顺序组合算子 ( ; )、选择组合算子 ( + ) 和并行组合算子 ( $|[A]|$ ). 设 Act 是给定的所有动作的集合，设 $\surd$ 表示成功终止的动作名. 进程的定义直接采用如下 BNF (backus naur form) 形式：

$$P ::= \ \mathbf{0} \mid \mathbf{1} \mid a \mid P;P \mid P+P \mid P|[A]|P$$

$$(a \in \mathrm{Act} \cup \{\surd\}, A \subseteq \mathrm{Act} \cup \{\surd\}\ ),$$

这里，构造进程的意义解释如下：

**0** 表示这个进程不做任何事情；

**1** 表示成功的终止，即执行了特殊的动作 $\surd$, 然后变成 **0** ；

$P_1;P_2$ 表示 $P_1$ 和 $P_2$ 这两个进程的顺序组合，进程 $P_1$ 成功地终止后才能执行进程 $P_2$；

$P_1+P_2$ 表示 $P_1$ 和 $P_2$ 这两个进程的选择组合，要么选择 $P_1$ 执行，要么选择 $P_2$ 执行；

$P_1|[A]|P_2$ 表示 $P_1$ 和 $P_2$ 这两个进程的并行组合，其中 $A$ 是同步动作集，进程 $P_1$ 和 $P_2$ 在执行到包含在动作集 A 的动作时必须保持同步执行，如果 $A=\emptyset$，则用 $||$ 表示 $|[\emptyset]|$，即 $||=|[\emptyset]|$.

规定算子的优先级为 + 比 ; 弱，但比 $|[A]|$ 强.

一般地，给出一种语言，首先需要定义其操作语义. 操作语义的基本思想是用抽象的方法描述语言中每一成分的执行效果，以免所描述的语义依赖于该语言实现时所用的具体计算机. 通常的做法是设计一个抽象机，定义一组抽象状态，把语言的语法表示成抽象的形式，然后指明抽象机每加工一个语言成分时将对状态作何种改变. 这种语义方法与语言实现的关系比较紧密，但是难以用数学的方法处理，而且对语义描述者个人使用的实现方法依赖很大.

1981 年，Plotkin[78] 提出了一种新的操作语义描述方法，称为结构化的操作语义. 其基本思想是：复合成分的操作语义应该可以归结为它的各个组成部分的操作语义. 这样，在证明语义正确性时就可以使用结构归纳法，因此，结构化操作语义的本质是把公理化方法引入操作语义之中.

Plotkin 定义了形式为 $\xrightarrow{a}$ 的进程变迁，$P \xrightarrow{a} P'$ 表示行为 $P$ 能执行动作 $a \in \mathrm{Act} \cup \{\surd\}$ 并进化成行为 $P'$. 上述语言 **L** 的结构化操作语义如下：

$$\frac{}{\mathbf{1} \xrightarrow{\surd} \mathbf{0}}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1;P_2 \xrightarrow{a} P_1';P_2}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1+P_2 \xrightarrow{a} P_1'}$$

$$\frac{P_2 \xrightarrow{a} P_2'}{P_1+P_2 \xrightarrow{a} P_2'}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1|[A]|P_2 \xrightarrow{a} P_1'|[A]|P_2} \quad (a \notin A)$$

$$\frac{P_2 \xrightarrow{a} P_2'}{P_1|[A]|P_2 \xrightarrow{a} P_1|[A]|P_2'} \quad (a \notin A)$$

$$\frac{P_1 \xrightarrow{a} P_1' \wedge P_2 \xrightarrow{a} P_2'}{P_1|[A]|P_2 \xrightarrow{a} P_1'|[A]|P_2} \quad (a \in A)$$

注意，事件结构通常用于进程代数表达式的指称语义模型. 本书中，为了简单和方便，例子中常常使用进程代数表示系统，这里给出进程代数的一些说明，目的是便于读者理解.

## 2.2 事件结构

在并发理论中，事件结构 (event structure) 是并发模型的主要分支，它最先用于连接 Petri 网和 Scott 域理论 [71]，后来扩展用于作为进程代数的语义模型 [13,61,92]. 当前，事件结构模型已经被广泛地延伸、扩展 [13]. 为简化问题，本书选用最基本的事件结构 (prime event structure)[72,94] 作为研究模型.

事件结构是非交织 (noninterleaving) 模型的一个主要分支 [42]. 其基本要素有：被标记的事件以及这些事件之间的因果关系、矛盾关系和独立关系. 构成事件结构的基本单元是事件. 一个事件是一个发生了的动作的模型化. 对于一个动作的发生，它所发生的时间、发生的原因以及所发生的环境都各不相同. 事件就是指具体的发生了的动作.

下面引入有关事件结构的基本概念，这些定义将在本章和其他章节有用.

假设 Act 是一个动作的集合.

**定义 2.2.1** 一个**基本事件结构** [94]( 在集合 Act 上 ) $\mathcal{E}$ 是一个四元组 $\langle E, <, \#, l\rangle$，其中，

- $E$ 是事件集合；
- $<\subseteq E \times E$ 是非自反的偏序关系，并满足“有限原因规则”：

  $\forall e \in E : \{d \in E \mid d < e\}$ 是有限的, 另外，$<$ 的逆用 $>$ 表示；

- $\# \subseteq E \times E$ 是非自反的有限的矛盾关系，并满足“矛盾继承规则”：

  $$\forall d, e, f \in E : d < e \wedge d\#f \Rightarrow e\#f;$$

- $l : E \longrightarrow \text{Act}$ 是动作标记函数.

基本事件结构 (本书中，有时简称“事件结构”) 用如下方式描述系统：动作名是系统执行的活动，一个被动作 $a(a \in \text{Act})$ 标记的事件表示动作 $a$ 一次特定的出现，$d < e$ 表示 $d$ 先于 $e$ 出现，$d\#e$ 表示在一次运行中 $d$ 和 $e$ 不会同时一起出现. 从上述定义也能得出 **独立关系 co** 的概念：$d \text{ co} e \Leftrightarrow \neg(d = e \vee d < e \vee e < d \vee d\#e)$,

也就是说，独立关系 co 是：$\text{co}=_{\text{def}}E \times E \setminus (< \cup > \cup \# \ \cup \{(e,e) \mid e \in E\})$. 很明显，co 是对称的. 根据上述定义，得出 $<$、$>$、$\#$ 和 co 是 $E \times E$ 的一个划分.

为下面叙述方便，给出一些补充定义. 设 $E$ 表示在动作集合 Act 上的**事件结构域**，设 $\emptyset$ 表示**空事件结构** $\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$. 一个事件结构 $\mathcal{E}$ 的四个组件一般被表示如下：$E_{\mathcal{E}}, <_{\mathcal{E}}, \#_{\mathcal{E}}$ 和 $l_{\mathcal{E}}$. 如果上下文不产生混淆，索引 $\mathcal{E}$ 将被省略. 一个事件结构 $\mathcal{E}$ 是空的当且仅当 $E_{\mathcal{E}}$ 是空集，是有限的当且仅当 $E_{\mathcal{E}}$ 是有限的，是无矛盾的当且仅当 $\#_{\mathcal{E}} = \emptyset$. 对于集合 $X \subseteq E_{\varepsilon}$, 事件结构 $\mathcal{E}$ 限制到集合 $X$ 被定义如下：$\mathcal{E}|_X = \langle X, < \cap (X \times X), \# \cap (X \times X), l|_X\rangle$.

两个事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 是**同构**的 (表示为 $\mathcal{E} \simeq_{\text{e}} \mathcal{F}$) 当且仅当在它们的事件集间存在一个双射并保持 $<$、$\#$ 和相同的标记. 除非特别说明，我们不区分同构的事件结构.

事件结构的行为一般通过**格局**(configuration) 来描述，格局是具有特定性质的事件集合. 格局也可以考虑是系统可能的状态. 下面给出它的定义.

**定义 2.2.2** 设集合 $X$ 是事件结构 $\mathcal{E}$ 的事件集 $E_{\mathcal{E}}$ 的一个子集.

(1) $X$ 是 **左封闭的** 当且仅当 $\forall d, e \in E$: $e \in X \wedge d < e \Rightarrow d \in X$；

(2) $X$ 是 **无矛盾的** 当且仅当 $\mathcal{E}|_X$ 是无矛盾的；

(3) $X$ 是一个 **格局** 当且仅当它既是左封闭的又是无矛盾的.

这里，用 $C(\mathcal{E})$ 表示事件结构 $\mathcal{E}$ 所有格局的集合. 一个格局 $X(X \in C(\mathcal{E}))$ 称作 (成功) 终止的格局当且仅当 $\forall d \in E$: $d \notin X \Rightarrow \exists e \in X$: $e\#d$.

在一个事件结构 $\mathcal{E} = \langle E, <, \#, l\rangle$ 中，一个 (在 Act 上) 标记偏序集，简称偏序集，是一个三元组 $(X, <, l)$，这里 $<$ 是在集合 $X(X \subseteq E)$ 上的原因关系并且 $l$: $X \longrightarrow \text{Act}$ 是标记函数. 对于任何 $X \subseteq E$ 是格局，由于矛盾关系为空，因此 $\mathcal{E}|_X = \langle X, < \cap (X \times X), \# \cap (X \times X), l|_X\rangle$ 能看成一个偏序集.

**定义 2.2.3** 设 $\mathcal{E}$ 是个事件结构.

(1) 设 $\mathcal{X} = (X, <_X, l_X)$ 和 $\mathcal{Y} = (Y, <_Y, l_Y)$ 是标记在 Act 上的两个标记偏序集. $\mathcal{X}$ 和 $\mathcal{Y}$ 是**同构的**(表示为 $\mathcal{X} \simeq_{\text{p}} \mathcal{Y}$) 当且仅当存在一个双射 $f$: $X \to Y$，使得 $\forall d, e \in X$：$d <_X e \Leftrightarrow f(d) <_Y f(e)$，并且 $l_X = l_Y \circ f$. 在集合 Act 上的标记偏序集的同构类称作偏序多集 (pomset)；

(2) 格局 $X(X \in C(\mathcal{E}))$ 的偏序多集是 $\text{pomset}(X) = [(X, <_X, l|_X)]_{\simeq_{\text{p}}}$. 事件结构 $\mathcal{E}$ 中的所有偏序多集组成的集合是 $\text{pomsets}(\mathcal{E}) = \{\text{pomset}(X) \mid X \in C(\mathcal{E})\}$.

事件结构经常用图形来表示 [41,72,94]. 在图形中，事件结构的原因关系用箭头表示，直接的矛盾关系仍用符号 $\#$ 表示，继承的矛盾关系不考虑，而独立关系不会被明显表示出来.

**例 2.2.1** 下面的图形表示了一个有三个事件 $e_a, e_b, e_c$(分别被动作 $a$, $b$, $c$ 标记) 的事件结构 $\varepsilon$. 事件 $e_a$ 是事件 $e_b$ 出现的原因，并与事件 $e_c$ 矛盾，根据矛盾继

承规则，事件 $e_c$ 与事件 $e_b$ 也矛盾. 事件结构 $\varepsilon$ 可能的格局有 $\emptyset, \{e_a\}, \{e_a, e_b\}$ 和 $\{e_c\}$，其中 $\{e_a, e_b\}$ 和 $\{e_c\}$ 是终止的.

$$\varepsilon: \begin{array}{l} e_a \longrightarrow e_b \\ \# \\ e_c \end{array}$$

事件结构通常用于定义进程代数的指称语义，下面将给出进程代数语言 **L** 的事件结构指称语义. 首先给出一些辅助定义. 假定存在一个无限的事件空间 $E_U$.

设进程 $P_i(i=1,2)$ 同态映射到事件结构 $\mathcal{E}_i(i=1,2)$，即 $\mathcal{E}[\![P_i]\!] = \mathcal{E}_i = \langle E_i, <_i, \#_i, l_i\rangle$，并设 $E_1 \cap E_2 = \emptyset$. 同时设事件结构 $\mathcal{E}$ 所有的初始事件构成的集合 $\text{init}(\mathcal{E}) = \{e \in E_{\mathcal{E}} \mid \nexists e' \in E_{\mathcal{E}} : e' < e\}$，设所有成功终止的事件组成的集合 $\text{exit}(\mathcal{E}) = \{e \in E_{\mathcal{E}} \mid l(e) = \surd\}$.

对于进程的并行组合操作算子 $|[A]|$ 需要特别说明，其中的同步事件存在并行组合的双方可以直接配对，不同步的事件则需要与特殊符号 $*$ $(* \notin E_U)$ 配对. 对于 $A \subseteq \text{Act}$，假定 $E_i^s = \{e \in E_i \mid l_i \in (\text{Act} \cup \{\surd\})\}$ 和 $E_i^f = E_i \backslash E_i^s$.

**定义 2.2.4**

$\mathcal{E}[\![\mathbf{0}]\!] = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$;

$\mathcal{E}[\![\mathbf{1}]\!] = \langle \{e_{\surd}\}, \emptyset, \emptyset, \{(e_{\surd}, \surd)\} \rangle$;

$\mathcal{E}[\![a]\!] = \langle \{(e_a\}, \emptyset, \emptyset, \{(e_a, a)\} \rangle$;

$\mathcal{E}[\![P_1; P_2]\!] = \langle E_1 \cup E_2, <, \#_1 \cup \#_2, l_1 \cup l_2 \rangle$, 其中 $< = <_1 \cup <_2 \cup \text{exit}(\mathcal{E}_1) \times \text{init}(\mathcal{E}_2)$;

$\mathcal{E}[\![P_1 + P_2]\!] = \langle E_1 \cup E_2, <_1 \cup <_2, \#, l_1 \cup l_2 \rangle$, 其中 $\# = \#_1 \cup \#_2 \cup (\text{init}(\mathcal{E}_1) \times \text{init}(\mathcal{E}_2)) \cup (\text{init}(\mathcal{E}_2) \times \text{init}(\mathcal{E}_1))$;

$\mathcal{E}[\![P_1 |[A]| P_2]\!] = \langle E, <, \#, l, \rangle$ 其中 $E = (E_1^f \times \{*\}) \cup (\{*\} \times E_2^f) \cup \{(e_1, e_2) \in E_1^s \times E_2^s | l_1(e_1) = l_2(e_2)\}$, $(e_1, e_2) < (e_1', e_2') \Leftrightarrow (e_1 <_1 e_1') \vee (e_2 <_2 e_2')$, $(e_1, e_2) \# (e_1', e_2') \Leftrightarrow (e_1 \#_1 e_1') \vee (e_2 \#_2 e_2')$, 和 $l((e_1,\ e_2)) = \text{if}_{e_1} = * \text{then } l_2(e_2) \text{ else } l_1(e_1)$.

由于事件结构的“矛盾继承规则”，用事件结构刻画某些包含有顺序组合和并行组合的进程时，可能需要复制事件或去掉一些事件，这样特殊处理后才能建立起事件结构模型. 下面给出两个例子.

**例 2.2.2** 一个进程 $(c+d); b$，如果用动作直接表示事件，那么可以说，$b$ 出现的原因是 $c$ 或 $d$. 根据事件结构的定义，$b$ 只可能有唯一的原因事件. 因此，该进程对应的事件结构如下图，这里的事件 $b$ 被复制了一份.

$$\begin{array}{l} c \longrightarrow b \\ \# \\ d \longrightarrow b \end{array}$$

下面给出一个有并行组合的例子.

**例 2.2.3** 一个进程 $(a+(c;a))|[\{a\}]|(a;b)$, 这里用动作直接表示事件, 根据事件结构的定义, $b$ 必须被复制一份, 所以该进程对应的事件结构为

$$\begin{array}{l} a \longrightarrow b \\ \# \\ c \longrightarrow a \longrightarrow b \end{array}$$

## 2.3 动作细化

根据上一章的介绍, 动作细化是在系统的基本构成单位是动作的框架下, 实现并发系统的层次化的一种基本操作, 其基本思想是将系统高层次的基本单位替换为低层次进程, 即首先用一组简洁、紧凑的动作集合来抽象描述一个简单的并发系统的行为, 然后在该层次上选定一个或几个抽象的未加解释的动作, 并分别用低一层次的具体的活动 (动作集合) 进行替换, 逐层做下去, 直至得到详尽的系统设计或实现. 这是动作细化的直观定义, 非形式化的. 然而, 建立在进程代数和一些模型上的动作细化的概念是形式化的、理论化的. 本书考察的模型是事件结构, 所以动作细化的概念建立在事件结构之上 [41]. 这个概念遵循以下四个性质:

(1) 动作不用解释, 不考虑细化的正确性问题;

(2) 一个给定的系统描述的细化由细化函数确定, 细化函数是将动作细化成系统描述;

(3) 所有动作都可能被细化;

(4) 在动作之间的所有原因 (独立) 和矛盾关系被细化后的系统继承.

**定义 2.3.1** 一个函数 $\mathrm{ref}:\mathrm{Act}\rightarrow E-\{\emptyset\}$ 称作 ( 对事件结构的 )**细化函数**, 当且仅当 $\forall a\in \mathrm{Act}: \mathrm{ref}(a)$ 是非空的、有限的和无矛盾的. 设事件结构 $\mathcal{E}\in E$, 那么 $\mathrm{ref}\ (\mathcal{E})$ 是一个事件结构, 其定义如下:

- $E_{\mathrm{ref}(\mathcal{E})}=\{(e,e')\mid e\in E_{\mathcal{E}}, e'\in E_{\mathrm{ref}(l_{\mathcal{E}}(e))}\}$;
- $(d,d')<_{\mathrm{ref}(\mathcal{E})}(e,e')$ 当且仅当 $d<_{\mathcal{E}} e$ 或者 $d=e\wedge d'<_{\mathrm{ref}(l_{\mathcal{E}}(d))} e'$;
- $(d,d')\ \#_{\mathrm{ref}(\mathcal{E})}\ (e,e')$ 当且仅当 $d\ \#_{\mathcal{E}}\ e$;
- $l_{\mathrm{ref}(\mathcal{E})}(e,e')=l_{\mathrm{ref}(l_{\mathcal{E}}(e))}(e')$.

根据定义 2.3.1 和定义 2.2.1, 直接有下列命题.

**命题 2.3.1**

(1) 如果事件结构 $\mathcal{E} \in E$ 并且 ref 是一个细化函数，那么 $\mathrm{ref}(\mathcal{E})$ 是一个事件结构；

(2) 如果两个事件结构 $\mathcal{E}, \mathcal{F} \in E$，并且 $\mathcal{E} \simeq_{\mathrm{e}} \mathcal{F}$，以及 ref 是一个细化函数，那么 $\mathrm{ref}(\mathcal{E}) \simeq_{\mathrm{e}} \mathrm{ref}(\mathcal{F})$.

有了上述定义，可以定义格局的细化.

**定义 2.3.2** 设事件结构 $\mathcal{E} \in \boldsymbol{E}$，并设 ref 是一个细化函数. 称 $\widetilde{X}$ 是**函数** ref **细化格局** $X(X \in C(\mathcal{E}))$ **后所得的格局** 当且仅当

- $\widetilde{X} = \bigcup\limits_{e \in X} \{e\} \times X_e$，其中 $\forall e \in X : X_e \in C(\mathrm{ref}(l_{\mathcal{E}}(e))) - \{\emptyset\}$,
- $e \in \mathrm{busy}(\widetilde{X}) \Rightarrow e$ 是 $X$ 中关于 $<_{\mathcal{E}}$ 最大的，其中 $\mathrm{busy}(\widetilde{X}) = \{e \in X \mid X_e$ 不是终止的 $\}$.

下面的命题阐明了细化后的事件结构与原事件结构在行为上的关系.

**命题 2.3.2** 设事件结构 $\mathcal{E} \in E$，并设 ref 是一个细化函数，那么 $C(\mathrm{ref}(\mathcal{E})) = \{\widetilde{X} \mid \widetilde{X}$ 是格局 $X(X \in C(\mathcal{E}))$ 被函数 ref 细化的格局 $\}$.

证明参看文献 [41] 的命题 2.2 和命题 3.3.

## 2.4 标记变迁系统

对于并发系统的行为刻画，标记变迁系统是一种传统的被广为接受的基于抽象状态机的状态变迁模型. 标记变迁系统首先分别由 Keller [58] 和 Lien [60] 两人独立提出来的，但现在一些作者往往认为是 Plotkin 首先提出来的 [78]. 这可能是因为 Keller 和 Lien 当时提出的变迁系统是由状态和变迁两个元组构成，而 Plotkin 则认为变迁系统是由状态、变迁和标记 (动作) 三个元组构成，这更接近现在普遍采用的 “标记变迁系统” 或 “状态变迁系统” . 目前变迁系统已向多方面扩展，对其名字有多种不同的称谓，但本质上都是基于上述三位学者的思想.

标记变迁系统 LTS(labelled transition system) 常用于形式化描述分布式系统的行为，成为这些系统说明、实现和测试研究的有力工具. 标记变迁系统还能作为许多形式描述语言的语义模型，如 CCS、CSP、LOTOS、ESTELLE、SDL 等. 另外，标记变迁系统是形式化一致性测试的一个重要研究方向. 一个标记变迁系统由状态以及带标号的状态间的变迁组成 [86].

**定义 2.4.1** 一个**标记变迁系统**是一个四元组 $\langle S, L, T, s_0 \rangle$，其中，

- $S$: 状态集合，

- $L$: 标记 (动作) 集合,
- $T \subseteq S \times L \times S$: 变迁关系,
- $s_0$: 初始状态.

注意，有时表示系统从状态 $s$ 执行动作 $a$ 后变迁到状态 $t$ 可以写作 $(s, a, t)$ 或 $s \xrightarrow{a} t$，另外可以用后继函数 succ 表示系统从某状态执行一个动作后的后继状态，即 $t = \mathrm{succ}(s, a)$. 把从某状态 $s$ 出发能执行的动作集定义为 $\mathrm{enabled}(s) = \{a \mid \exists t : s \xrightarrow{a} t\}$. 系统的一个执行序列是由动作序列构成，如果动作序列 $\overline{a} = a_1 a_2 \cdots a_n (n > 0)$ 表示该变迁系统中存在执行序列 $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} t$，可以简写成 $s_0 \stackrel{\overline{a}}{\Longrightarrow} t$，也可以用后继函数表示 $t = \mathrm{succ}(s_0, \overline{a})$. 有时候称 $\overline{a}$ 是该变迁系统的一个**迹**. $|\overline{a}|$ 表示动作序列 $\overline{a}$ 所有动作的集合，即 $|\overline{a}| = \{a_1, a_2, \cdots, a_n\}$. 本书中 $L^*$ 表示在一个标记变迁系统中基于动作集 $L$ 的所有非空的动作序列.

标记变迁系统也常常用图形来表示，一般用空心圆圈或实心圆圈表示系统的状态，用带箭头的线表示变迁关系，其中没有箭头的线头指向变迁前的状态，有箭头的线头指向变迁进化到达的状态. 动作标记标注在用带箭头的线上面，如图 2.1. 但是，由于考虑表示方便，表示状态的圆可能被其他代替，本书后面的章节将会看到，请注意理解.

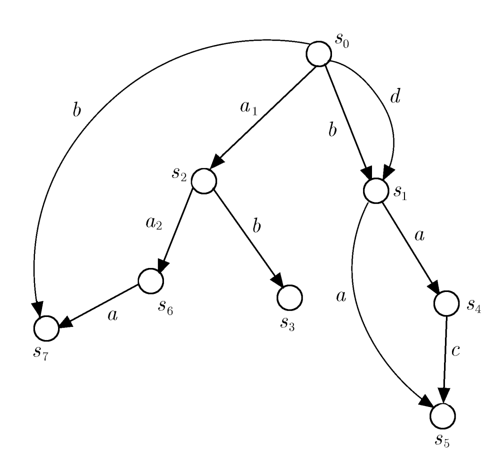



图 2.1　一个标记变迁系统

两个标记变迁系统 $\mathrm{LTS}_1 = \langle S_1, L_1, T_1, s_{0_1} \rangle$ 和 $\mathrm{LTS}_2 = \langle S_2, L_2, T_2, s_{0_2} \rangle$ 是同构的，表示为 $\mathrm{LTS}_1 \simeq_l \mathrm{LTS}_2$，当且仅当在它们的状态集间存在一个双射 $f : S_1 \to S_2$，使得 $\forall s, t \in S_1, \forall a \in L_1 : s \xrightarrow{a} t \Leftrightarrow f(s) \xrightarrow{a} f(t)$.

事实上，一个事件结构的行为可由变迁系统来展示，其中每个格局是一个状态，标记集合是标记事件的动作集合，变迁是每两个格局之间的单个动作变迁，以及初始格局空集 (∅) 是初始状态. 具体关系请参看后面的章节.

## 2.5 模型检验

近年来模型检验方法发展很快，人们提出了许多改进方法，但基本思想还是没有改变. 一般认为，模型检验是用时态逻辑公式表达系统 (程序或电路) 的所期望的性质，用有限状态机 FSM(finite state machine) 表示系统的状态转移结构，通过遍历有限状态机来检验时态逻辑公式的正确性. 如果不能检验公式的正确性，系统将根据遍历的路径给出一个反例，使用户发现公式不成立的原因，帮助用户改正错误. 在模型检验中，表达系统的所期望的性质可以采用多种逻辑，不同的逻辑，其实现算法有很大差异，在偏序约简实现的模型检验中，一般采用线性时序逻辑 LTL[34] 公式表示系统的性质. 下面给出线性时序逻辑 LTL 公式的语法和语义 [22].

如果给定一个原子命题集合 AP，时态运算符 G(global)，F(future)，X(next)，U(until)，线性时序逻辑 LTL 公式语法可以归纳定义为

- AP 中的每个元素是一个 LTL 公式；
- 如果 $\varphi$ 和 $\psi$ 是 LTL 公式，则 $\neg\varphi, \varphi\wedge\psi, \mathrm{X}\varphi, \varphi\mathrm{U}\psi$ 也是 LTL 公式；
- 设 $A\wedge\neg A$ 简写为 false，true 是 ¬false 的简写，则

  $$\varphi\vee\psi=\neg((\neg\varphi\wedge\neg\psi)),\ \mathrm{F}\varphi=\mathrm{trueU}\varphi,\ \mathrm{G}\varphi=\neg\mathrm{F}\neg\varphi.$$

可以用建立在原子命题集合上的无限字符串 $\omega=x_0x_1\cdots$ 对线性时序逻辑 LTL 公式解释，即自然数到其幂集的一个映射. 在模型检验中，一般用标记变迁系统作 LTL 公式的语义. 用 $\omega_i$ 表示在 $\omega$ 中从下标 $i$ 开始的字符串，LTL 公式的语义可以归纳如下：

- $\omega\vDash p$ 当且仅当 $p\in x_0$，$p$ 是 AP 中的原子命题；
- $\omega\vDash\neg\varphi$ 当且仅当 $\omega\vDash\varphi$ 不成立；
- $\omega\vDash\varphi\wedge\psi$ 当且仅当 $\omega\vDash\varphi$ 并且 $\omega\vDash\psi$；
- $\omega\vDash X\varphi$ 当且仅当 $\omega_1\vDash\varphi$ ；
- $\omega\vDash\varphi U\psi$ 当且仅当 $\exists i\geq 0$ 满足 $\omega_i\vDash\varphi$，并且 $\forall j(0\leq j<i)$ 满足 $\omega_j\vDash\varphi$.

给定一个标记变迁系统 $M$ 和一个 LTL 公式 $\varphi$，模型检验就是对每一条从 $x_0$ 开始的路径 $\omega$，检验 $\omega \vDash \varphi$ 是否成立. 如果都成立，记为 $M \vDash \varphi$.

模型检验实现起来是非常简单的，但是随着系统规模的增大，刻画它们的状态变迁系统的状态数目将呈指数增加而引起组合爆炸，这就是状态爆炸问题，一旦这种情况出现，就限制了模型检验的应用. 因此必须采用其他方法缓解状态爆炸问题，这方面的研究已经取得了丰硕的成果，但仍需要继续研究. 本书中研究的对称约简和后面改进的偏序约简都是为此服务的.

# 第三章　进程代数中的对称性

## 3.1 引　　言

进程代数是一种形式化描述复杂并发系统的建模工具，是一种高层的描述语言，是支持并发分布系统的组合描述及其性质形式化证明的代数语言. 它以代数形式来描述模型，并为模型化的系统定义了一套完整的语法和语义.

任何硬件和软件并发系统都存在大量的相同或同构的组件，相应地，如果用进程代数来刻画它们，必然会存在以进程代数表示出的相同或同构的组件，这些组件就是系统对称性的体现，对称性结构在系统分析中常常是不必要的，增加了分析系统的难度. 特别是在复杂的并发系统分析中去掉对称中的多余的组件，可以节约大量的处理时间，并能增加对系统分析和理解的正确性. 进程代数已经被广泛用于刻画并发系统，指导并发系统的设计与开发，所以研究它的本身约简问题是非常必要的. 一般的研究工作从行为上研究其约简方法，其中主要通过一些行为等价来进行约简，这些研究有必要，也是很好的方法，但是从结构方面来研究进程代数的约简问题几乎没有. 尽管 Hermanns 和 Ribaudo[50] 已经做了一些工作，但他们主要针对并行算子在随机进程代数中的对称问题的处理，本质上仍是行为方面的研究，这与我们的研究是完全不同的. 本章将进程代数看成是一种代数结构，基于这种代数结构建立起进程代数的对称性概念，并建立了约简算法，同时给出一些例子说明对称性理论的正确性. 这些工作扩展了我们以前的工作 [55].

## 3.2 进程代数与自同构

### 3.2.1 进程代数

上一章已经给出了采用 BNF 形式定义的进程代数语法. 这里，为了处理问题方便，我们给出进程代数的代数结构定义，其思想来源于 Bergstra 和 Klop 的进程代数 ACP[9,5].

在给出进程代数的代数结构定义之前，先定义一个概念. 在 2.1 节中，由算子; (顺序组合)、+(选择组合) 和 $||[A]||$(并行组合) 等运算符形成的进程代数语言里，把每个字称作一个 **进程项**. 显然，最小的进程项就是动作.

设 $\mathbb{P}$ 是所有进程项的集合 (字母表 Act 是所有动作的集合，在 2.1 节中已有定义.)

**定义 3.2.1**(进程) 一个**进程** $\mathcal{P}$ 是一个四元组 $\langle P,\ ;,\ \dagger,\ ||\rangle$, 其中,

- $P \subseteq \mathbb{P}$, 有限的进程项集合, 这里考虑最小的进程项 (即字母表 Act 中的动作);
- $; \subseteq P \times P$, 非自反的偏序关系 ( **顺序组合关系**);
- $\dagger \subseteq P \times P$, 非自反的对称关系 ( **选择组合关系**) 并满足**选择组合继承规则**: $\forall p, q, r \in P:\ (p;q) \wedge (p\dagger r) \Rightarrow (q\dagger r)$;
- $|| \subseteq P \times P$, 对称关系 (**并行组合关系**).

这里, 进程 $\mathcal{P}$ 的组件表示为: $P_{\mathcal{P}}$, $;_{\mathcal{P}}$ , $\dagger_{\mathcal{P}}$ 和 $||_{\mathcal{P}}$, 如果不引起误会, 下标 $\mathcal{P}$ 将被删掉. 进程 $\mathcal{P}$ 是**空**的当且仅当集合 $P_{\mathcal{P}}$ 是空集, $\mathcal{P}$ 是**有限**的当且仅当集合 $P_{\mathcal{P}}$ 是有限集合, $\mathcal{P}$ 是**无矛盾**的当且仅当 $\dagger_{\mathcal{P}} = \emptyset$. 对集合 $X \in P_{\mathcal{P}}$, 进程 $\mathcal{P}$ **限制到集合** $X$ **上**定义为 $\mathcal{P}|_X = \langle X,\ ;\cap(X\times X),\ \dagger\cap(X\times X),\ ||\cap(X\times X)\rangle$.

注意, 上述定义 3.2.1 中的元组 $P$ 实际上满足 $P \subseteq$ Act. 为了区分相同的动作, 将给出不同的进程项名. 根据进程代数 ACP 的思想, 进程代数语言满足如下公理:

对于任意的 $x, y, z \in \mathbb{P}$, 有

$$\begin{aligned}
PA1:&\quad x + y = y + x,\\
PA2:&\quad x + (y + z) = (x + y) + z,\\
PA3:&\quad x + x = x,\\
PA4:&\quad x;(y;z) = (x;y);z,\\
PA5:&\quad x||[A]||y = y||[A]||x.
\end{aligned}$$

定义 3.2.1 有利于研究进程代数的对称性, 然而它比 BNF 范式表示进程复杂. 在本书中, 这两种方式表示进程都将采用. 显然, 它们存在对应关系. 这里, 我们给出一些例子. 空进程 $\mathbf{0}$ 的结构表示是 $\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$. 终止进程 $\mathbf{1}$ 不表示成结构形式, $\mathbf{1}$ 即是动作 $\surd$. 只有一个动作的进程 $a$ 则是 $\langle\{a,\surd\},\{(a,\surd)\},\emptyset,\emptyset\rangle$. 本书有时不考虑终止动作. 因此, 进程 $a;(b||c)$ 的结构是 $\langle\{a,b,c\},\{(a,b),(a,c)\},\emptyset,\{(b,c),(c,b)\}\rangle$, 其中的进程项被认为是动作自己, 而进程 $a||a$ 由于存在相同的动作, 所以其结构是 $\langle\{p_1,p_2\},\emptyset,\emptyset,\{(p_1,p_2),(p_2,p_1)\}\rangle$, 其中进程项 $p_1, p_2$: $p_1 = p_2 = a$.

接下来定义两进程同构. 两进程 $\mathcal{P},\mathcal{Q}$ **同构**(表示为 $\mathcal{P}\cong\mathcal{Q}$) 当且仅当存在一个双射 $f: P_{\mathcal{P}} \to P_{\mathcal{Q}}$ , 使得对任意 $p, q \in P_{\mathcal{P}}$: $p;_{\mathcal{P}}\, q \Longleftrightarrow f(p);_{\mathcal{Q}} f(q)$, $p\ \dagger_{\mathcal{P}}\ q \Longleftrightarrow f(p)\dagger_{\mathcal{Q}} f(q)$, $p\ ||_{\mathcal{P}}\ q \Longleftrightarrow f(p)||_{\mathcal{Q}} f(q)$ , 并且进程项 $p$ 和 $q$ 对应相同的动作, 即 $p = q$.

### 3.2.2 自同构

本小节将给出定义进程对称性的一些预备知识. 一般地, 置换群的定义是基于对称群给出的. 因此首先给出对称群的定义.

**定义 3.2.2**(对称群) 设 $X$ 是一个有限集合, 集合 $S := \{f : X \to X \mid f$ 是一个双射 $\}$ 和其中的映射复合作为二元操作构成的群称为集合 $X$ 上的**对称群**, 表示

为 $S_X$. 其中，任何一个双射 $f$ $(f \in S)$ 称为一个置换，表示为 $f \in S_X$.

接下来定义置换群.

**定义 3.2.3**(置换群) 设 $X$ 是一个有限集合，并设 $S_X$ 是集合 $X$ 上的对称群. 对称群 $S_X$ 的任何子群称为集合 $X$ 上的一个**置换群**.

现在，我们知道了在一个集合上的置换群是由该集合自己映射到自己的双射和这些双射的复合作为二元操作构成的. 显然，对称群是一个特殊的置换群. 在一个集合上的置换群，它有许多好的性质，其中之一就是它能诱导出一个等价关系 [83].

**命题 3.2.1** 设 $X$ 是一个有限集合，并设 $G$ 是集合 $X$ 上的置换群. 如果在集合 $X$ 上的关系 $R$ ，满足 $R = \{(a, b) \mid f(a) = b, f \in G\}$, 则 $R$ 是一个等价关系.

上述命题很容易证明，此处省略.

一个集合上的等价关系的等价类形成该集合的一个划分. 因此，对于集合 $X$ ，如果在该集合上存在一个置换群，则该置换群能诱导出集合 $X$ 的一个划分. 本书中，置换群就是用于划分一个进程中进程项, 以便我们用进程项的等价类 (进程项的等价类有时称为进程项的轨道 (orbit)) 来研究进程的对称性.

**定义 3.2.4**(自同构) 设进程 $\mathcal{P} = \langle P, \ ;, \ \dagger, \ ||\rangle$，并设 $G$ 是进程 $\mathcal{P}$ 的进程项集合 $P_{\mathcal{P}}$ 上的一个置换群. 一个置换 $f \in G$ 称作进程 $\mathcal{P}$ 的**自同构**当且仅当 $f$ 满足如下条件：$\forall p_1, p_2 \in P_{\mathcal{P}}$ :

- $p_1 ; p_2 \Rightarrow f(p_1) ; f(p_2)$;
- $p_1 \dagger p_2 \Rightarrow f(p_1) \dagger f(p_2)$;
- $p_1 \mid\mid p_2 \Rightarrow f(p_1) \mid\mid f(p_2)$.

一个置换群 $G$ 称作进程 $\mathcal{P}$ 的**自同构群**当且仅当每一个置换 $f \in G$ 是进程 $\mathcal{P}$ 的自同构. 注意，因为每一个置换 $f \in G$ 都有一个逆，它也是一个自同构，所以能证明，$f \in G$ 是进程 $\mathcal{P}$ 的一个自同构当且仅当 $f$ 满足如下条件：$\forall p_1, p_2 \in P_{\mathcal{P}}$ :

- $p_1 ; p_2 \Leftrightarrow f(p_1) ; f(p_2)$;
- $p_1 \dagger p_2 \Leftrightarrow f(p_1) \dagger f(p_2)$;
- $p_1 \mid\mid p_2 \Leftrightarrow f(p_1) \mid\mid f(p_2)$.

因此，很容易理解, 如果置换群 $G$ 中的每个置换都是进程 $\mathcal{P}$ 的自同构，则群 $G$ 就是进程 $\mathcal{P}$ 的自同构群.

## 3.3 进程代数的对称性

设 $G$ 是进程项集合 P 上的一个置换群，并设进程项 $p \in P$，则 $p$ 的**轨道**是集

合 $\theta(p)=\{q \mid \exists f \in G: f(p)=q\}$. 从每个轨道 $\theta(p)$ 选一个代表，表示为 $\mathrm{rep}(\theta(p))$. 显然，对称性有如下性质：$p$ ($p \in \mathrm{P}$) 与代表 $\mathrm{rep}(\theta(p))$(即 $\theta(p)$ 中的任何一个) 必须展示一致的行为和相同的性质. 也就是说，这些进程项是相同的动作：

$$p=\mathrm{rep}(\theta(p)).$$

因此，直观地说，对称约简得到的商模型就是合并一个轨道中的所有进程项，并把它看成这些进程项的一个代表. 这里给出如下定义.

**定义 3.3.1**(商进程结构) 设进程 $\mathcal{P}=\langle P,\ ;,\ \dagger,\ ||\rangle$，并设 $G$ 是进程 $\mathcal{P}$ 的进程项集合 $P_{\mathcal{P}}$ 上的一个自同构群. **商进程结构** $\mathcal{P}_G=\langle P_G,\ ;_G,\ \dagger_G,\ ||_G\rangle$ 定义如下：

- $P_G=\{\theta(p) \mid p \in P\}$ 是进程 P 的进程项的轨道的集合；
- 顺序组合关系 $;_G$ ：$;_G=\{(\theta(p_1),\theta(p_2)) \mid (p_1,p_2)\in ;\}$；
- 并行组合关系 $||_G$ ：$||_G=\{(\theta(p_1),\theta(p_2))|(p_1,p_2)\in ||\}$；
- 选择组合关系 $\dagger_G$ ：$\dagger_G = P_G \times P_G \setminus (\ ||_G \cup ;_G \cup \overline{;||}_G\ )$，这里，$\overline{;||}_G = \{(\theta(p_1),\theta(p_2)) \mid \overline{;||}=P\times P\setminus(; \cup ||),(p_1,p_2)\in\overline{;||}\}$.

注意，因为 $G$ 是一个自同构群，所以 $;_G$，$\dagger_G$ 和 $||_G$ 是明确的，并且独立于选择的代表. 从上述定义可以看出，一个进程具有对称性表明其对称的子进程能完整地代替整个进程的行为，而且，我们使用 3.2.1 节给出的公理 $P$A1、$P$A2、$P$A3、$P$A4 和 $P$A5 能推导出这些对称的子进程在结构上也是一致的. 这里给出一个例子，它是 3.6 节的第一个例子. 有兴趣的读者可以先看看.

接下来讨论商进程结构的一些性质. 下面给出的命题将表明商进程结构仍是一个进程，并且它涉及的动作集合与原进程的动作集合是相同的，同时该命题也将表明商进程结构保留了原进程的顺序和并行组合关系，但选择组合关系可能被约简.

**命题 3.3.1** 设进程 $\mathcal{P}=\langle \mathrm{P},\ ;,\ \dagger,\ ||\rangle$，其中 $\mathrm{Act}_{\mathcal{P}}(\mathrm{Act}_{\mathcal{P}}\subseteq \mathrm{Act})$ 构成进程 $\mathcal{P}$ 的动作集合. 并设 $\mathcal{P}_G=\langle P_G,\ ;_G,\ \dagger_G,\ ||_G\rangle$ 是进程 $\mathcal{P}$ 的商进程结构，同时设集合 $\mathrm{Act}_{\mathcal{P}_G}$ 是构成商进程结构的动作集合. 则有

(1) $\mathcal{P}_G$ 是一个进程；

(2) $\mathrm{Act}_{\mathcal{P}_G}=\mathrm{Act}_{\mathcal{P}}$；

(3) 顺序和并行组合关系在对称约简下被保留，即 $\forall p,q \in P: p\ ;\ q \Rightarrow \theta(p)\ ;_G\,\theta(q)$ 和 $p\ ||\ q \Rightarrow \theta(p)\ ||_G\theta(q)$; 然而, 选择组合关系可能被约简，即 $\forall p,q\in P: p \dagger q \not\Rightarrow \theta(p)\ \dagger_G\theta(q)$.

证明根据定义直接可得，此处省略.

在例 3.6.1 中，很容易测试商进程结构的上述性质. 下面将讨论它的其他性质.

**命题 3.3.2** 设进程 $\mathcal{P}=\langle P,\ ;,\ \dagger,\ ||\rangle$，并设 $G$ 是进程 $\mathcal{P}$ 的一个自同构群.

(1) 如果进程项 $p \in P_{\mathcal{P}}$，并且 $|\theta(p)| > 1$ ( $|\theta(p)|$ 表示集合 $\theta(p)$ 中元素的数目)，则 $\forall q \in \theta(p) : q \neq p \Rightarrow \theta(p) = \theta(q)$.

(2) 如果进程项 $p \in P_{\mathcal{P}}$，并且 $q \in \theta(p)(p \neq q)$，则存在一个双射 $g : P_{\mathcal{P}} \setminus \{p, q\} \longrightarrow P_{\mathcal{P}} \setminus \{p, q\}$，使得对任意进程项 $r \in P \setminus \{p, q\}$:

- $p \,;r \Leftrightarrow (q \,; g(r)) \wedge (g(r) = r)$;
- $r \,; p \Leftrightarrow (g(r) \,; q) \wedge (g(r) = r)$;
- $r \dagger p \Leftrightarrow (g(r) \dagger q) \wedge (g(r) = r)$;
- $r || p \Leftrightarrow (g(r) || q) \wedge (g(r) = r)$.

**证明** (1) 根据定义直接可证，此处省略.

(2) 因为存在一个自同构 $f \in G$, 所以有 $p \Box r \Leftrightarrow f(p) \Box f(r)$，其中 $\Box \in \{;, \dagger, ||\}$，进而有 $f(p) \in \theta(p)$. 由于 $p = f(p)$ 蕴含这个结论是显然的，因此考虑 $p \neq f(p)$. 现在，设 $q = f(p)$, 并设 $g = f|_{P \setminus \{p,q\}}$. 因此有 $p \Box r \Leftrightarrow q \Box g(r)$，其中 $\Box \in \{;, \dagger, ||\}$. 根据对称性，则有 $r \Box p \Leftrightarrow g(r) \Box q$，其中 $\Box \in \{;, \dagger, ||\}$. 所以，结论成立.

在一个给定的进程里，命题 3.3.2(1) 表明了在进程项的轨道中所有的进程项都是相同的动作，而命题 3.3.2(2) 表明在进程项的轨道中两个不同的进程项与进程中的其他进程项有相同的顺序、并行和选择关系. 第二个性质将被用于研究后面的对称约简算法，但这里不准备进一步讨论这个问题. 我们必须先研究商进程结构是否真的保留了原进程的所有行为，即行为等价在对称约简下的保持问题.

## 3.4 行为等价的保持

本节将讨论基于进程代数定义的两类等价：交织迹等价 (interleaving trace equivalence) 和交织互模拟等价 (interleaving bisimulation equivalence)，有时也直接称迹等价和互模拟等价. 在线性时间与分支时间等价关系谱 [36] 中，交织迹等价是最粗的等价，而除了树等价 (tree equivalence)，交织互模拟等价是最细的行为等价，也是最著名的、最好的等价，因此我们选用这两类等价来研究对称约简的保持问题. 就是说，我们将研究商进程结构与原进程是否存在交织迹等价和交织互模拟等价. 注意，这里的交织迹等价和交织互模拟等价是基于进程代数建立起来的等价，与本书后面部分以事件结构作为模型建立起来的交织迹等价和交织互模拟等价尽管本质上是一致的，但存在一些差别. 因此需要分别定义它们.

为了叙述方便，先给出一些预备定义. 给定一个进程 $\mathcal{P} = \langle P, \,;, \,\dagger, \,||\rangle$，对于集合 $X \subseteq P$，设 $\dagger_{\mathcal{P}}(X) = \{p \in P \mid \exists q \in X : (q, p) \in \dagger\}$，定义

$$P^{'} =_{\text{def}} P \setminus \dagger_{\mathcal{P}}(X),$$

和

$$\mathcal{P} \searrow X =_{\text{def}} \langle P^{'},\ ;\cap(P^{'} \times P^{'}),\ \dagger \cap (P^{'} \times P^{'}),\ || \cap (P^{'} \times P^{'})\rangle.$$

**定义 3.4.1**(变迁) 设进程 $\mathcal{P} = \langle P_{\mathcal{P}},\ ;_{\mathcal{P}},\ \dagger_{\mathcal{P}},\ ||_{\mathcal{P}}\rangle$, 并设进程 $\mathcal{Q} = \langle \mathrm{Q}_{\mathcal{Q}},\ ;_{\mathcal{Q}},\ \dagger_{\mathcal{Q}},\ ||_{\mathcal{Q}}\rangle$. $\mathcal{P} \xrightarrow{a} \mathcal{Q}$ 称作一个**变迁**当且仅当 $a \in \mathrm{Act}$ ，并且 $\exists p \in P_{\mathcal{P}} : p = a, \mathcal{Q} = \mathcal{P} \searrow \{p\}$.

显然, 这里的进程被看成状态. 事实上, 对于任何进程 $\mathcal{P}$, 其行为也可以用一个标记变迁系统来展示, 即初始状态是进程本身, 终止状态是进程 $\surd$, 其状态是执行了一些动作的剩余 (进程). 这种处理方式与进程代数的操作语义 [78] 一致, 比如进程代数 LOTOS [11]. 这里用 AS($\mathcal{P}$) 表示进程 $\mathcal{P}$ 所有变迁形成的进程 (状态).

在下面的部分里, 将研究商进程结构和原进程之间的行为关系.

**命题 3.4.1** 设进程 $\mathcal{P} = \langle P,\ ;,\ \dagger,\ ||\rangle$, 并设 $\mathcal{P}_G = \langle P_G,\ ;_G,\ \dagger_G,\ ||_G\rangle$ 是进程 $\mathcal{P}$ 的商进程结构, 则

(1) $\forall p \in P, \mathcal{X}, \mathcal{X}' \in \mathrm{AS}(\mathcal{P}) : \mathcal{X}' = \mathcal{X} \setminus \{p\} \Rightarrow \exists \mathcal{X}_G, \mathcal{X}'_G \in \mathrm{AS}(\mathcal{P}_G) : \mathcal{X}'_G = \mathcal{X}_G \setminus \{\theta(p)\}$;

(2) $\forall q \in P_G, \mathcal{X}_G, \mathcal{X}'_G \in \mathrm{AS}(\mathcal{P}_G) : \mathcal{X}'_G = \mathcal{X}_G \setminus \{q\} \Rightarrow \exists p \in P, \exists \mathcal{X}, \mathcal{X}' \in \mathrm{AS}(\mathcal{P}) : \mathcal{X}' = \mathcal{X} \setminus \{p\} \wedge (p = \mathrm{rep}(q))$.

**证明** (1) 假定 $\mathcal{X}' = \mathcal{X} \setminus \{p\} \nRightarrow \mathcal{X}'_G = \mathcal{X}_G \setminus \{\theta(p)\}$, 则有 $p \neq \mathrm{rep}(\theta(p))$, 这与定义 3.3.1 矛盾, 因此, 结论成立.

(2) 遵循上面 (1) 的证明方法, 结论很容易得出.

命题 3.4.1 表明在商进程结构中的每个状态都对应着原进程中的一个状态, 反之也成立. 下面接着讨论它们的变迁关系.

**命题 3.4.2** 设进程 $\mathcal{P} = \langle P,\ ;,\ \dagger,\ ||\rangle$, 并设 $\mathcal{P}_G = \langle P_G,\ ;_G,\ \dagger_G,\ ||_G\rangle$ 是进程 $\mathcal{P}$ 的商进程结构, 则

(1) $\forall \mathcal{X}, \mathcal{X}' \in \mathrm{AS}(\mathcal{P}), a \in \mathrm{Act} : \mathcal{X} \xrightarrow{a}_{\mathcal{P}} \mathcal{X}' \Rightarrow \exists \mathcal{Y}, \mathcal{Y}' \in \mathrm{AS}(\mathcal{P}_G) : \mathcal{Y} \xrightarrow{a}_{\mathcal{P}_G} \mathcal{Y}'$;

(2) $\forall \mathcal{Y}, \mathcal{Y}' \in \mathrm{AS}(\mathcal{P}_G), a \in \mathrm{Act} : \mathcal{Y} \xrightarrow{a}_{\mathcal{P}_G} \mathcal{Y}' \Rightarrow \exists \mathcal{X}, \mathcal{X}' \in \mathrm{AS}(\mathcal{P}) : \mathcal{X} \xrightarrow{a}_{\mathcal{P}} \mathcal{X}'$.

**证明** (1) 对于变迁 $\mathcal{X} \xrightarrow{a}_{\mathcal{P}} \mathcal{X}'$, 根据定义 3.4.1, 则有 $\mathcal{X}' = \mathcal{X} \setminus \{p\}$ ，其中 $p = a$. 因为命题 3.4.1(1), 所以存在 $p \in P$ ，使得 $\mathcal{X}'_G = \mathcal{X}_G \setminus \{q\}$ , 其中 $\mathcal{X}_G, \mathcal{X}'_G \in \mathrm{AS}(\mathcal{P}_G), \mathrm{rep}(q) = p$, 即存在变迁 $\mathcal{X}_G \xrightarrow{a}_{\mathcal{P}_G} \mathcal{X}'_G$. 因此可令 $\mathcal{X}_G = \mathcal{Y}$ , 并且令 $\mathcal{X}'_G = \mathcal{Y}'$. 所以结论成立.

(2) 由于命题 3.4.1(2), 根据上面 (1) 结论, 该结论与其是对称的, 结论显然成立.

前面虽然讨论了若干问题, 但是为了引入交织迹等价, 还必须先给出迹的定义.

**定义 3.4.2**(迹) 一个字 $w = a_1 \cdots a_n \in \mathrm{Act}^*$ 称作一个进程 $\mathcal{P}$ 的**迹**当且仅当 $\exists \mathcal{X}_0, \cdots, \mathcal{X}_n \in \mathrm{AS}(\mathcal{P}) : \mathcal{X}_0 = \mathcal{P}$ 且 $\mathcal{X}_{i-1} \xrightarrow{a_i} \mathcal{X}_i (i = 1, \cdots, n)$.

这里，用 $\mathrm{tr}s(\mathcal{P})$ 表示进程 $\mathcal{P}$ 的所有迹的集合. 接下来，定义交织迹等价.

**定义 3.4.3**(交织迹等价) 设 $\mathrm{tr}s(\mathcal{P})$ 和 $\mathrm{tr}s(\mathcal{Q})$ 分别是进程 $\mathcal{P}$ 和 $\mathcal{Q}$ 的所有迹的集合. 进程 $\mathcal{P}$ 与进程 $\mathcal{Q}$ 称作**交织迹等价**(表示为 $\mathcal{P} \approx_{\mathrm{it}} \mathcal{Q}$) 当且仅当 $\mathrm{tr}s(\mathcal{P}) = \mathrm{tr}s(\mathcal{Q})$.

下面的定理表明商进程结构与原进程是交织迹等价的.

**定理 3.4.1** 设进程 $\mathcal{P} = \langle P_{\mathcal{P}}, ;_{\mathcal{P}}, \dagger_{\mathcal{P}}, ||_{\mathcal{P}} \rangle$，并设 $\mathcal{P}_G = \langle P_G, ;_G, \dagger_G, ||_G \rangle$ 是进程 $\mathcal{P}$ 的商进程结构，则 $\mathcal{P}_G \approx_{\mathrm{it}} \mathcal{P}$.

**证明** 假设任何迹 $w = a_1 \cdots a_n \in \mathrm{tr}s(\mathcal{P})$，其中，对动作 $a_i (i = 1, \cdots, n)$，则有 $p_i \in P_{\mathcal{P}} (i = 1, \cdots, n)$ 和 $p_i = a_i$. 根据定义 3.3.1，则有进程项 $\theta(p_i) \in P_G (i = 1, \cdots, n)$ ，其中 $\mathrm{rep}(\theta(p_i)) = p_i = a_i$，进而有 $w \in \mathrm{tr}s(\mathcal{P}_G)$. 利用对称性，对于任何 $v \in \mathrm{tr}s(\mathcal{P}_G)$，也有相应的迹 $v \in \mathrm{tr}s(\mathcal{P})$. 接下来证明它们迹的数目是相等的，即 $|\mathrm{tr}s(\mathcal{P})| = |\mathrm{tr}s(\mathcal{Q})|$. 假设 $|\mathrm{tr}s(\mathcal{P})| \neq |\mathrm{tr}s(\mathcal{Q})|$. 不失一般性，设 $|\mathrm{tr}s(\mathcal{P})| > |\mathrm{tr}s(\mathcal{Q})|$，则存在一个迹 $w' \in \mathrm{tr}s(\mathcal{P})$ 和 $w' \notin \mathrm{tr}s(\mathcal{Q})$，这与上面的证明相矛盾. 因此，有 $\mathrm{tr}s(\mathcal{P}) = \mathrm{tr}s(\mathcal{P}_G)$，即 $\mathcal{P}_G \approx_{\mathrm{it}} \mathcal{P}$.

有了上面的一些叙述和定义，现在能给出交织互模拟等价的定义了.

**定义 3.4.4**(交织互模拟等价) 设 $\mathcal{P}, \mathcal{Q}$ 是两个进程. 一个关系 $R \subseteq \mathrm{AS}(\mathcal{P}) \times \mathrm{AS}(\mathcal{Q})$ 称作进程 $\mathcal{P}$ 和 $\mathcal{Q}$ 之间的一个**交织互模拟**，当且仅当 $(\mathcal{P}, \mathcal{Q}) \in R$，并且如果 $(\mathcal{X}, \mathcal{Y}) \in R$，则

- $\mathcal{X} \xrightarrow{a}_{\mathcal{P}} \mathcal{X}',\ a \in \mathrm{Act} \ \Rightarrow\ \exists \mathcal{Y}' : \mathcal{Y} \xrightarrow{a}_{\mathcal{Q}} \mathcal{Y}' \wedge (\mathcal{X}', \mathcal{Y}') \in R$,
- $\mathcal{Y} \xrightarrow{a}_{\mathcal{Q}} \mathcal{Y}',\ a \in \mathrm{Act} \ \Rightarrow\ \exists \mathcal{X}' : \mathcal{X} \xrightarrow{a}_{\mathcal{P}} \mathcal{X}' \wedge (\mathcal{X}', \mathcal{Y}') \in R$.

两进程 $\mathcal{P}$ 和 $\mathcal{Q}$ 称作**交织互模拟等价**(表示为 $\mathcal{P} \approx_{\mathrm{ib}} \mathcal{Q}$)，当且仅当进程 $\mathcal{P}$ 和 $\mathcal{Q}$ 之间存在一个交织互模拟.

建立了交织互模拟的概念，现在考虑商进程结构与原进程是否交织互模拟等价，下面给出一个定理.

**定理 3.4.2** 设进程 $\mathcal{P} = \langle P_{\mathcal{P}}, ;_{\mathcal{P}}, \dagger_{\mathcal{P}}, ||_{\mathcal{P}} \rangle$，并设 $\mathcal{P}_G = \langle P_G, ;_G, \dagger_G, ||_G \rangle$ 是进程 $\mathcal{P}$ 的商进程结构，则 $\mathcal{P}_G \approx_{\mathrm{ib}} \mathcal{P}$.

**证明** 必须在进程 $\mathcal{P}$ 和 $\mathcal{P}_G$ 之间构造出一个交织互模拟. 设关系 $\widetilde{R} \subseteq \mathrm{AS}(\mathcal{P}) \times \mathrm{AS}(\mathcal{P}_G)$. 用归纳法构造关系 $\widetilde{R}$. 首先，让 $(\mathcal{P}, \mathcal{P}_G)$ 属于关系 $\widetilde{R}$. 设 $\mathcal{X} \xrightarrow{a}_{\mathcal{P}} \mathcal{X}' (a \in \mathrm{Act})$，因为命题 3.4.2(1)，则 $\exists (\mathcal{Y} \xrightarrow{a}_{\mathcal{P}_G} \mathcal{Y}')$. 又因为命题 3.4.2(2)，则有 $\forall (\mathcal{Y} \xrightarrow{a}_{\mathcal{P}_G} \mathcal{Y}') \Rightarrow \exists (\mathcal{X} \xrightarrow{a}_{\mathcal{P}} \mathcal{X}')$. 现在，令 $(\mathcal{X}, \mathcal{Y})$ 属于 $\widetilde{R}$，接着令 $(\mathcal{X}', \mathcal{Y}')$ 属于 $\widetilde{R}$. 最后，令所有从 $\mathcal{X}'$ 和 $\mathcal{Y}'$ 分别变迁得到的后续的状态，将它们对应的状态作成配对都加入 $\widetilde{R}$ 中. 显然，$\widetilde{R}$ 是一个交织互模拟. 因此，$\mathcal{P}_G \approx_{\mathrm{ib}} \mathcal{P}$.

以上定理充分说明，商进程结构与原进程在行为上存在等价关系，商进程结构保留了原进程的行为，也表明我们定义的进程的对称性存在一定合理性.

## 3.5 一个约简算法

本节将研究进程代数语言的对称约简算法. 为了方便叙述，先给出一个定义.

**定义 3.5.1** 设 $\mathcal{P}$ 是一个进程. 对于任何进程项 $p \in P_{\mathcal{P}}$，进程项 $p$ 的一个**关系结构** 是一个四元组 $R_p = \langle\ P^p, ;^p\ ,\ \dagger^p,\ ||^p\rangle$, 其中

- $P^p$ 是与进程项 $p$ 存在关系的所有进程项；
- $;^p$ 是进程项 $p$ 与其他进程项的顺序组合关系；
- $\dagger^p$ 是进程项 $p$ 与其他进程项的选择组合关系；
- $||^p$ 是进程项 $p$ 与其他进程项的并行组合关系.

两个进程项 $p, q$ 称作**结构相同** (表示为 $p \asymp q$) 当且仅当 $p\dagger q$, $p = q$, 并且两结构关系 $R_p$ 和 $R_q$ 存在一个双射：$f : P^p \to P^q$ ，使得对任意 $r \in P^p$：$p;^p r \Leftrightarrow q;^q f(r)$, $p\dagger^p r \Leftrightarrow q\dagger^q f(r)$, $p||^p r \Leftrightarrow q||^q f(r)$ ，并且 $f(r) = r$.

**命题 3.5.1** 设进程 $\mathcal{P} = \langle P,\ ;,\ \dagger,\ ||\rangle$, 并设 $G$ 是进程 $\mathcal{P}$ 的一个自同构群. 对于任意两进程项 $p, p \in P_{\mathcal{P}}$，如果 $p \asymp q$，则有 $\theta(p) = \theta(q)$.

**证明** 根据定义 3.5.1，如果 $p \asymp q$，则进程项 $p, q$ 相互替换后，进程 $\mathcal{P}$ 的所有性质处在同构级别上 ( up to isomorphism). 因此，我们能构造一个自同构 $f$ 使 $p$ 映射到 $q$，其他的进程项自己映射到自己. 根据定义 3.2.4，则 $f \in G$. 所以有 $\theta(p) = \theta(q)$.

有了上面的准备工作，约简算法很容易建立.

**算法 3.5.1**

输入: 进程 $\mathcal{P} = \langle P,\ ;,\ \dagger,\ ||\rangle$

输出: 进程结构 $\langle P_G,\ ;_G\ ,\ \dagger_G,\ ||_G\rangle$

第一步:

$\mathcal{P}_G := \langle\emptyset, \emptyset, \emptyset, \emptyset\rangle$

$\theta_{p_i} := \{p_i\}\ (p_i \in P_{\mathcal{P}})$

第二步:

$P_0 := P_{\mathcal{P}}$ ($P_0$ 是临时变量)

**If** $P_0 = \emptyset$ **then** stop

**Repeat**

从集合 $P_0$ 选出任意一个进程项 $p_i$

$P_0 := P_0 \setminus \{p_i\}$

$P_t := P_0$ ($P_t$ 是临时变量)

**Repeat**

从集合 $P_t$ 选出任意一个进程项 $p_j$

**If** $p_i \asymp p_j$ **then**

$\{\ \theta_{p_i} := \theta_{p_i} \cup \{p_j\}$

$\theta_{p_j} := \theta_{p_j} \cup \{p_i\}$

$P_0 := P_0 \setminus \{p_j\}\ \}$

Let $P_t := P_t \setminus \{p_j\}$

**Until** $P_t = \emptyset$

**Until** $P_0 = \emptyset$

第三步:

$P_G := \{\theta_{p_i} \mid p_i \in P_{\mathcal{P}}\}$

$;_G := \{(\theta_{p_i}, \theta_{p_j}) \mid \theta_{p_i} \neq \theta_{p_j}, (p_i, p_j) \in ;_{\mathcal{P}}\} \cup \{(\theta_{p_j}, \theta_{p_i}) \mid \theta_{p_j} \neq \theta_{p_i}, (p_j, p_i) \in ;_{\mathcal{P}}\}$

$\dagger_G := \{(\theta_{p_i}, \theta_{p_j}) \mid \theta_{p_i} \neq \theta_{p_j}, (p_i, p_j) \in \dagger_{\mathcal{P}}\} \cup \{(\theta_{p_j}, \theta_{p_i}) \mid \theta_{p_j} \neq \theta_{p_i}, (p_j, p_i) \in \dagger_{\mathcal{P}}\}$

$||_G := \{(\theta_{p_i}, \theta_{p_j}) | \theta_{p_i} \neq \theta_{p_j}, (p_i, p_j) \in ||_{\mathcal{P}}\} \cup \{(\theta_{p_j}, \theta_{p_i}) \mid \theta_{p_j} \neq \theta_{p_i}, (p_j, p_i) \in ||_{\mathcal{P}}\}$

$\mathcal{P}_G := \langle P_G,\ ;_G,\ \dagger_G,\ ||_G \rangle$

**Return** $\langle P_G,\ ;_G,\ \dagger_G,\ ||_G \rangle$

上述算法显然是正确的，并且在有限步终止，其时间复杂度主要依赖进程项集合 $P_{\mathcal{P}}$ 的元素数目，主要时间花费在计算 $R_{p_i}(p_i \in P_{\mathcal{P}})$ 上，所以时间复杂度为 $O(|P_{\mathcal{P}}|^2)$.

## 3.6 例　子

这里将给出两个例子，读者结合上面的定义和算法很容易理解这些例子. 首先看一个简单的例子.

**例 3.6.1** 进程 $P = (a||b)+(b||a)+(c;d)$ 对应结构 $\mathcal{P}$，其中有六进程项：$p_1, p_2, p_3, p_4, p_5, p_6, p_1 = p_2 = a, p_3 = p_4 = b, p_5 = c, p_6 = d$，并行关系有 $p_1||p_3, p_4||p_2$，顺序关系有 $p_5;p_6$，选择关系有 $p_1, p_2$ 中的每个进程项分别与 $p_3, p_4$ 中每个进程项存在选择关系，并且 $p_1, p_2, p_3, p_4$ 中的每个进程项也与 $p_5, p_6$ 中的每个进程项存在选择关系. 进程 $P$ 实际上是 $(p_1||p_3)+(p_4||p_2)+(p_5;p_6)$. 注意，根据公理 $P$A5 有 $(a||b) = (b||a)$. 设

$\mathcal{P}_1 = \langle \{p_3, \surd\}, \{(p_3, \surd)\}, \emptyset, \emptyset \rangle,$

$\mathcal{P}_2 = \langle \{p_4, \surd\}, \{(p_4, \surd)\}, \emptyset, \emptyset \rangle,$

$\mathcal{P}_3 = \langle \{p_1, \surd\}, \{(p_1, \surd)\}, \emptyset, \emptyset \rangle,$

$\mathcal{P}_4 = \langle \{p_2, \surd\}, \{(p_2, \surd)\}, \emptyset, \emptyset \rangle,$

$\mathcal{P}_5 = \langle \{p_6, \surd\}, \{(p_6, \surd)\}, \emptyset, \emptyset \rangle.$

因此，原进程 $\mathcal{P}$ 的状态和变迁关系如图 3.1 所示.

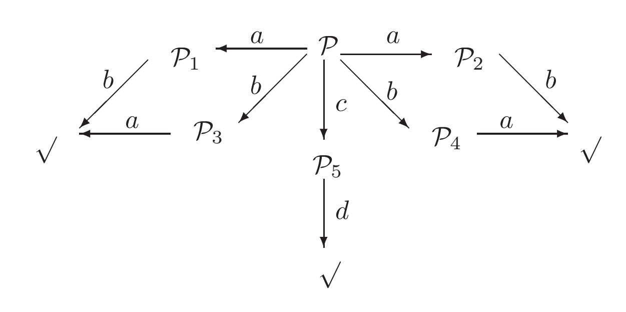



图 3.1

显然，进程项 $p_1$ 与其他进程项存在的关系如下：顺序组合关系是空集 $\emptyset$，选择组合关系是集合 $\{(p_1, p_2),\ (p_1, p_4),\ (p_1, p_5), (p_1, p_6),\ (p_2, p_1), (p_4, p_1), (p_5, p_1), (p_6, p_1)\}$，以及并行组合关系是集合 $\{(p_1, p_3), (p_3, p_1)\}$. 而进程项 $p_2$ 与其他进程的关系如下：顺序组合关系是空集 $\emptyset$，选择组合关系是集合 $\{(p_2,\ p_1),\ (p_2,\ p_3),\ (p_2,\ p_5), (p_2, p_6), (p_1, p_2), (p_3, p_2), (p_5, p_2), (p_6, p_2)\}$，以及并行组合关系是集合 $\{(p_2,\ p_4),\ (p_4, p_2)\}$. 由于 $p_1 = p_2 = a$, $p_3 = p_4 = b$，则有 $q_1 \asymp q_2$. 用同样的方法，也可以得出 $p_3 \asymp p_4$. 这里，$\mathrm{P}_{\mathcal{P}} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$，我们能在集合 $P_{\mathcal{P}}$ 上构造一个置换群 $G$，仅有两个置换如下：

$$\begin{pmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{pmatrix}, \qquad \begin{pmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ p_2 & p_1 & p_4 & p_3 & p_5 & p_6 \end{pmatrix}.$$

群 $G$ 明显是一个自同构群, 因此, 我们知道进程项的轨道: $\theta(p_1) = \theta(p_2) = \{p_1, p_2\}$, $\theta(p_3) = \theta(p_4) = \{p_3, p_4\}$，$\theta(p_5) = \{p_5\}$ 和 $\theta(p_6) = \{p_6\}$. 综上所述，进程 $\mathcal{P}$ 的商进程结构为 $\mathcal{P}_G = \langle P_G,\ ;_G,\ \dagger_G,\ ||_G\rangle$：

- $P_G = \{\ \{p_1, p_2\}, \{p_3, p_4\}, \{p_5\}, \{p_6\}\ \}$;
- $;_G = \{\ (\{p_5\}, \{p_6\})\ \}$;
- $\dagger_G = \{\ (\{p_1, p_2\}, \{p_5\}),\ (\{p_5\}, \{p_1, p_2\}), (\{p_3, p_4\}, \{p_5\}), (\{p_5\}, \{p_3, p_4\}), (\{p_1, p_2\}, \{p_6\}), (\{p_6\}, \{p_1, p_2\}), (\{p_3, p_4\}, \{p_6\}), (\{p_6\}, \{p_3, p_4\})\ \}$;
- $||_G = \{\ (\{p_1, p_2\}, \{p_3, p_4\}),\ (\{p_3, p_4\}, \{p_1, p_2\})\ \}$.

因此，$\mathrm{rep}(\{p_1, p_2\}) = p_1 = p_2 = a$, $\mathrm{rep}(\{p_3, p_4\}) = p_3 = p_4 = b$, $\mathrm{rep}(\{p_5\}) = p_5 = c$, $\mathrm{rep}(\{p_6\}) = p_6 = d$. 这时候，我们明白了商进程结构 $\mathcal{P}_G$ 实际是进程 $(a||b) + (c; d)$ 或进程 $(b||a) + (c; d)$. 图 3.2 描述了商进程结构 $\mathcal{P}_G$ 的行为. 显然，商进程结构 $\mathcal{P}_G$ 比原进程 $\mathcal{P}$ 简单.

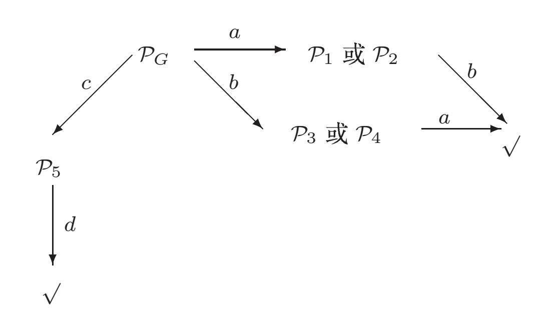



图 3.2

接下来看另外一个非常有趣的例子，它表明对称约简和一些吸收律 [14] 存在一致性. 此外，这个例子将直接验证命题 3.3.2(2) 所得出的结论.

**例 3.6.2** 进程 $Q = (b||(a+c)) + (a||b) + ((b+c)||a)$. 为了区分相同的动作名的不同出现，设 $q_1 = q_2 = q_3 = a, q_4 = q_5 = q_6 = b$，$q_7 = q_8 = c$，则有 $Q = (q_4||(q_1+q_7)) + (q_2||q_5) + ((q_6+q_8)||q_3)$，它对应结构 $\mathcal{Q} = \langle P_{\mathcal{Q}}, ;_{\mathcal{Q}}, \dagger_{\mathcal{Q}}, ||_{\mathcal{Q}}\rangle$：

- $P_{\mathcal{Q}} = \{\ q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\ \}$;
- $;_{\mathcal{Q}} = \emptyset$;
- $\dagger_{\mathcal{Q}} = \{\ (q_4,q_2),(q_4,q_5),(q_4,q_6),(q_4,q_8),(q_4,q_3),(q_2,q_4),(q_5,q_4),(q_6,q_4),(q_8,q_4),(q_3,q_4),(q_1,q_7),(q_1,q_2),(q_1,q_5),(q_1,q_6),(q_1,q_8),(q_1,q_3),(q_7,q_1),(q_2,q_1),(q_5,q_1),(q_6,q_1),(q_8,q_1),(q_3,q_1),(q_7,q_2),(q_7,q_5),(q_7,q_6),(q_7,q_8),(q_7,q_3),(q_2,q_7),(q_5,q_7),(q_6,q_7),(q_8,q_7),(q_3,q_7),(q_2,q_6),(q_2,q_8),(q_2,q_3),(q_6,q_2),(q_8,q_2),(q_3,q_2),(q_5,q_6),(q_5,q_8),(q_5,q_3),(q_6,q_5),(q_8,q_5),(q_3,q_5),(q_6,q_8),(q_8,q_6)\ \}$;
- $||_{\mathcal{Q}} = \{\ (q_4,q_1),(q_1,q_4),(q_4,q_7),(q_7,q_4),(q_2,q_5),(q_5,q_2),(q_6,q_3),(q_3,q_6),(q_8,q_3),(q_3,q_8)\ \}$.

这里考虑进程以交织方式执行并发动作，并设

$$\mathcal{Q}_1 = \langle\{q_1, q_7, \surd\}, \{(q_1,\surd),(q_7,\surd)\}, \emptyset, \emptyset\rangle,$$
$$\mathcal{Q}_2 = \langle\{q_6, q_8, \surd\}, \{(q_4,\surd)\}, \emptyset, \emptyset\rangle,$$
$$\mathcal{Q}_3 = \langle\{q_4, \surd\}, \{(q_4,\surd)\}, \emptyset, \emptyset\rangle,$$
$$\mathcal{Q}_4 = \langle\{q_3, \surd\}, \{(q_3,\surd)\}, \emptyset, \emptyset\rangle,$$
$$\mathcal{Q}_5 = \langle\{q_4, \surd\}, \{(q_4,\surd)\}, \emptyset, \emptyset\rangle,$$
$$\mathcal{Q}_6 = \langle\{q_3, \surd\}, \{(q_3,\surd)\}, \emptyset, \emptyset\rangle,$$
$$\mathcal{Q}_7 = \langle\{q_5, \surd\}, \{(q_5,\surd)\}, \emptyset, \emptyset\rangle,$$
$$\mathcal{Q}_8 = \langle\{q_2, \surd\}, \{(q_2,\surd)\}, \emptyset, \emptyset\rangle.$$

图 3.3 展示了进程 $\mathcal{Q}$ 以交织方式执行并发动作的状态和变迁关系.

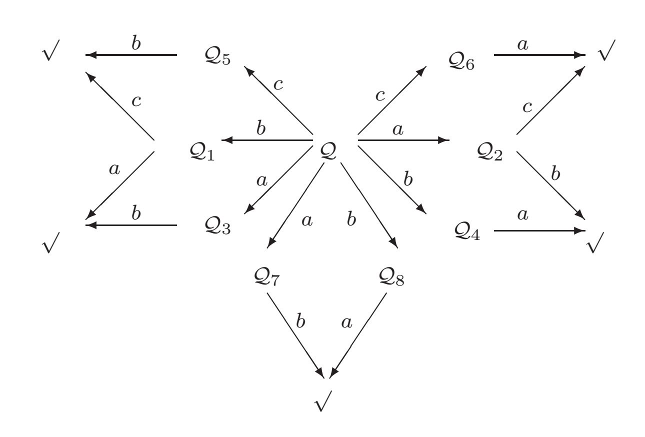



图 3.3　进程 $\mathcal{Q}$ 的状态和变迁关系

根据上面的叙述, 能够得出, 进程项 $q_1$ 与其他进程项的关系有: 顺序组合关系是空集 $\emptyset$, 选择组合关系为 $\{ (q_1,q_7),(q_1,q_2),(q_1,q_5),(q_1,q_6),(q_1,q_8),(q_1,q_3),(q_7,q_1),(q_2,q_1),(q_5,q_1),(q_6,q_1),(q_8,q_1),(q_3,q_1) \}$ 和并行组合关系是 $\{(q_4,q_1),(q_1,q_4) \}$; 而进程项 $q_2$ 与其他进程项的关系有: 顺序组合关系是空集 $\emptyset$, 选择组合关系为 $\{ (q_2,q_7), (q_1,q_2), (q_2,q_5), (q_2,q_6), (q_2,q_8), (q_2,q_3), (q_7,q_2), (q_2,q_1), (q_5,q_2), (q_6,q_2), (q_8,q_2), (q_3,q_2) \}$ 和并行组合关系是 $\{(q_5,q_1),(q_1,q_5) \}$. 因为 $q_4=q_5=b$, 则有 $q_1 \asymp q_2$. 同样地, 有 $q_5 \asymp q_6$. 因此, 存在具有下面两个置换的置换群 $G$:

$$\begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 \\ q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 \end{pmatrix}, \quad \begin{pmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 \\ q_2 & q_1 & q_3 & q_4 & q_6 & q_5 & q_7 & q_8 \end{pmatrix}.$$

根据前面商进程结构的定义和对称约简算法, 进程 $\mathcal{Q}$ 的商进程结构 $\mathcal{Q}_G = \langle P_G,\ ;_G,\ \dagger_G,\ ||_G \rangle$, 其中,

- $P_G = \{\{q_1,q_2\},\{q_3\},\{q_4\},\{q_5,q_6\},\{q_7\},\{q_8\}\}$;
- $;_G = \emptyset$;
- $\dagger_G = \{ (\{q_4\},\{q_5,q_6\}),(\{q_4\},\{q_8\}),(\{q_4\},\{q_3\}),(\{q_5,q_6\},\{q_4\}),(\{q_8\},\{q_4\}),(\{q_3\},\{q_4\}),(\{q_1,q_2\},\{q_7\}),(\{q_1,q_2\},\{q_5,q_6\}),(\{q_1,q_2\},\{q_8\}),(\{q_1,q_2\},\{q_3\}),(\{q_7\},\{q_1,q_2\}),(\{q_5,q_6\},\{q_1,q_2\}),(\{q_8\},\{q_1,q_2\}),(\{q_3\},\{q_1,q_2\}),(\{q_7\},\{q_5,q_6\}),(\{q_7\},\{q_8\}),(\{q_7\},\{q_3\}),(\{q_5,q_6\},\{q_7\}),(\{q_8\},\{q_7\}),(\{q_3\},\{q_7\}),(\{q_5,q_6\},\{q_8\}),(\{q_8\},\{q_5,q_6\}) \}$;

- $||_G = \{(\{q_1, q_2\}, \{q_4\}), (\{q_4\}, \{q_1, q_2\}), (\{q_4\}, \{q_7\}), (\{q_7\}, \{q_4\}), (\{q_5, q_6\}, \{q_3\}), (\{q_3\}, \{q_5, q_6\}), (\{q_8\}, \{q_3\}), (\{q_3\}, \{q_8\})\}$.

根据定义 3.3.1，显然有 $\text{rep}(\{q_1, q_2\}) = q_1 = q_2$ 和 $\text{rep}(\{q_5, q_6\}) = q_5 = q_6$. 图 3.4 描述了商进程结构 $\mathcal{Q}_G$ 的状态和变迁关系. 结合图 3.3，明显可以看出商进程结构 $\mathcal{Q}_G$ 与原进程 $\mathcal{Q}$ 存在较大差异. 事实上，商进程结构 $\mathcal{Q}_G$ 就是进程 $(b||(a+c)) + ((b+c)||a)$, 是原进程 $\mathcal{Q}$ 中的子进程 $(a||b)$ 被另外子进程 $(b||(a+c)) + (b+c)||a)$ 吸收掉而得出的.

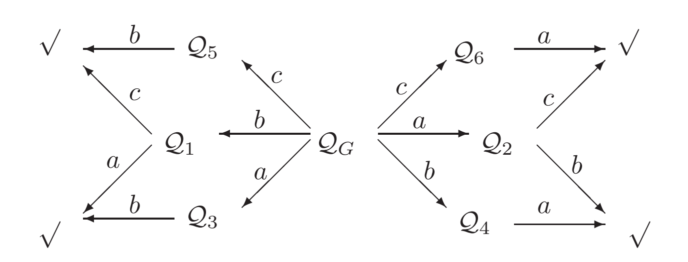



图 3.4 商进程结构 $\mathcal{Q}_G$ 的状态和变迁关系

上述例 3.6.2 表明对称约简不仅能约简一些相同或同构的组件，而且能约简一些非同构的组件. 这样研究似乎非常有意义，应该继续下去.

## 3.7 小　结

本章首先建立了进程代数中进程的对称性概念，即存在 (非平凡) 自同构的进程就具有对称性，其次，提出了求解对称约简所获得的进程 (称作商进程结构) 的方法，并且证明了商进程结构与原进程是交织迹和交织互模拟等价的，最后给出了对称约简算法和两个例子，说明了对称约简的必要性和意义. 本章没有考虑处理进程递归的情况，这是为了简化问题，实际上，递归算子的引入不会影响上述结论，但对于递归算子引入结构进程的结构定义，非常困难，必须要回到 BNF 形式的定义上，这方面的进一步工作是个有趣的课题.

很明显，对称约简是最细的行为约简，也是最细的结构 (静态) 约简. 对于一个给定的复杂系统，可以根据不同的需要建立不同大小的约简模型，从而为系统的设计、分析和开发服务. 但是，这些被约简得到的模型一般是行为模型，由于动态行为非常复杂，约简问题很可能变得困难. 相对来说，静态约简可能不考虑系统的动态行为，操作起来更容易. 对于进程代数，应该把这里所得到的方法和结果应用于实际问题中，不断完善基于进程代数语言的结构约简方法和理论.

# 第四章　事件结构模型的对称性

## 4.1　引　　言

对称在软件和硬件等并发系统中经常存在，比如在内存、高速缓存、注册表文件、网络协议中存在大量相同或同构的结构. 目前，人们已经在模型检验中针对对称约简进行了研究 [21,32]，这些约简技术建立在观察的基础之上，系统具有对称性则意味着存在保持状态标记和变迁关系的非平凡置换群. 这个置换群被用于定义系统状态空间上的等价关系，该等价关系诱导出的商模型通常比原模型规模要小，而且互模拟等价于原模型. 因此，商模型能够用于验证原模型使用例如 CTL* 公式表示的所有性质 [20].

在并发理论中，事件结构是并发模型的主要分支，它最先用于连接 Petri 网和 Scott 域理论 [71]，后来扩展用于作为进程代数的语义模型 [13,61,92]. 通过使用偏序约简，事件结构已经在模型检验中被采用 [77]. 事件结构无论被用在哪里，它都可能表现出对称性. 目前几乎没人注意到对称约简对一些等价关系的影响，比如在事件结构上的交织互模拟等价或偏序多集等价，特别值得一提的是，对称约简是否影响动作细化也没有得到考虑，而基于自上而下设计方法的动作细化已经在进程代数 [1, 3, 29, 49, 62, 63, 81, 96] 及其语义模型 [28, 33, 41, 64∼66, 88, 89] 中获得了人们非常广泛的关注.

在模型检验中，人们主要用 Kripke 结构作为模型考查其对称性. Kripke 结构是基于标记变迁系统的交织语义模型，不同于作为真并发模型的事件结构. 而事件结构的行为却是通过相应的标记变迁系统来表现. 因此，模型检验中的对称性不同于事件结构中的对称性. 当前，事件结构模型已经被广泛地延伸和扩展 [13]. 为简化问题，本书选用最基本的事件结构 (prime event structure)[72,94] 进行讨论.

我们认为，事件结构具有对称性意味着存在保持事件标记和保持事件间所有原因或独立关系的非平凡的置换群，该群能用于在这事件结构的事件集上定义一个等价关系. 通过该等价关系引入了事件结构的商事件结构模型，并证明了商结构与原事件结构是迹 (trace)、互模拟 (bisimulation) 和偏序多集迹 (pomset trace) 等价的. 而且也表明在动作细化下商事件结构与原事件结构具有相同的行为特性，也就是说，一个原事件结构的格局 (configuration) 和它对应的商事件结构的格局在动作细化下是同构的. 同时研究了商事件结构和原事件结构间在动作细化下迹、互模拟和偏序多集等价的保持问题.

相关工作如下：Emerson[32] 和 Clarke[21] 等人研究了在模型检验中的对称约简，他们是以 Kripke 结构作为模型，不同于我们的事件结构;Starke[84] 研究了 Petri 网

的对称性, 但本质上是基于变迁系统进行处理的;Hermanns 和 Ribaudo [50] 研究进程代数的对称性, 主要处理并行运算的相同组件. 这些研究完全不同于我们的研究.

# 4.2 事件结构中的对称

本节先引入置换群, 接着给出事件结构的自同构概念, 在此基础上定义了商事件结构. 实际上, 一个事件结构中存在自同构, 就称这个事件结构具有对称性, 可以通过对称约简方法进行事件结构约简.

## 4.2.1 置换群

先给出置换群的定义 [83].

**定义 4.2.1** 设 $X$ 是一个有限集合, 在集合 $X$ 上的一个双射函数: $X \to X$ 叫做一个置换. 在集合 $X$ 上的若干个置换作成的一个群叫做一个 **置换群** .

这里, 在集合 $X$ 上的一个置换群是由 $X$ 到 $X$ 的若干双射构成的集合及其双射的组合作为二元操作形成的群. 特别地, 置换群有一个很好的性质: 它能诱导出一个等价关系.

一般地, 一个集合上的等价关系确定该集合的一个划分. 因此, 对于有限的集合 $X$, 如果存在一个在集合 $X$ 上的置换群, 则这个置换群能诱导出 $X$ 的一个划分. 在本书中, 置换群就是用来划分一个事件结构的事件集, 以便我们用事件的等价类来研究事件结构的对称性问题.

## 4.2.2 自同构群

**定义 4.2.2** 设事件结构 $\mathcal{E} = \langle E, <, \#, l \rangle \in E$, 并设 $G$ 是在事件集合 $E$ 上的一个置换群. 一个置换 $f \in G$ 称作事件结构 $\mathcal{E}$ 的一个**自同构**当且仅当 $f$ 满足如下条件: 对于所有的 $e_1, e_1 \in E$:

- $e_1 < e_2 \Rightarrow f(e_1) < f(e_2)$;
- $e_1 \# e_2 \Rightarrow f(e_1) \# f(e_2)$;
- $e_1 \text{ co } e_2 \Rightarrow f(e_1) \text{ co } f(e_2)$;
- $l(e_1) = l(f(e_2))$.

一个置换群 $G$ 称作事件结构 $\mathcal{E}$ 的一个**自同构群**, 当且仅当每个置换 $f \in G$ 是 $\mathcal{E}$ 的一个自同构. 注意, 因为每个置换 $f \in G$ 都有一个逆元, 它也是一个自同构, 所以根据自同构群定义可以得出如下结论: $f \in G$ 是事件结构 $\mathcal{E}$ 的一个自同构当且仅当 $f$ 满足如下条件: 对于所有的 $e_1, e_2 \in E$:

- $e_1 < e_2 \Leftrightarrow f(e_1) < f(e_2)$;
- $e_1 \# e_2 \Leftrightarrow f(e_1) \# f(e_2)$;
- $e_1 \text{ co } e_2 \Leftrightarrow f(e_1) \text{ co } f(e_2)$;
- $l(e_1) = l(f(e_2))$.

### 4.2.3 商事件结构

设 $G$ 是在集合 $E$ 上的一个置换群，并设 $e \in E$，那么 $e$ 的轨道 (orbit) 是集合 $\theta(e) = \{d \mid \exists f \in G : f(e) = d\}$. 从每个轨道 $\theta(e)$ 选出一个代表表示为 $\text{rep}(\theta(e))$. 直观地说，我们的商模型是将事件合并到轨道中形成的模型.

**定义 4.2.3**(商事件结构) 设事件结构 $\mathcal{E} = \langle E, <, \#, l \rangle$，并设 $G$ 是事件结构 $\mathcal{E}$ 的事件集合 $E$ 上的一个自同构群，**商事件结构** $\mathcal{E}_G = \langle E_G, <_G, \#_G, l_G \rangle$ 定义如下：

- 集合 $E_G = \{\theta(e) \mid e \in E\}$ 是事件集合 $E$ 中事件的轨道的集合 $E$;
- $<_G = \{(\theta(e_1), \theta(e_2)) \mid (e_1, e_2) \in <\}$，并设 $<_G$ 的逆表示为 $>_G$;
- 独立关系 $\text{co}_G = \{(\theta(e_1), \theta(e_2)) \mid (e_1, e_2) \in \text{co}\}$;
- 矛盾关系 $\#_G = E_G \times E_G \setminus (<_G \cup >_G \cup \text{co}_G \cup \{(e'', e'') \mid e'' \in E_G\})$;
- 标记函数 $l_G(\theta(e)) = l(\text{rep}(\theta(e)))$.

注意，根据定义 4.2.2，下面将给出另一个定义. 事件结构 $\mathcal{E} = \langle E, <, \#, l \rangle$ 的一个自同构群 $G$ 是对动作 $a \in \text{Act}$ 的一个**不变群**，当且仅当下列条件成立：

$$\forall f \in G, \forall e \in E : l(e) = a \Leftrightarrow l(f(e)) = a.$$

称 $a$ 是在 $G$ 下的一个**不变量**. 因此，如果 $G$ 是对动作集合 Act 的所有动作的一个不变群，并且 $\mathcal{E}_G$ 是事件结构 $\mathcal{E}$ 的商事件结构，那么有如下结论：

$$\forall a \in \text{Act}, \forall e \in E : l(e) = a \Leftrightarrow l_G(\theta(e)) = l(\text{rep}(\theta(e))) = a.$$

**例 4.2.1** 进程 $P = (a||b) + (a||b) + (c; d)$ 对应的事件结构 $\mathcal{E}_P$，它有六个事件 $e_1, e_2, e_3, e_4, e_5, e_6$，并且 $l(e_1) = l(e_2) = a, l(e_3) = l(e_4) = b, l(e_5) = c, l(e_6) = d$，这里存在 $e_5 < e_6$ ，并且 $e_1, e_2$ 中的每个事件都与 $e_3, e_4$ 中的每个事件矛盾，同时 $e_1, e_2, e_3$ 和 $e_4$ 中的每个事件都和事件 $e_5$ 矛盾，根据矛盾继承规则，也和 $e_6$ 矛盾. 因此 $E_{\mathcal{E}_P} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$，我们能在集合 $E_{\mathcal{E}_P}$ 上构造一个置换群 $G$，其置换如下：

$$\begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{pmatrix}, \qquad \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ e_2 & e_1 & e_4 & e_3 & e_5 & e_6 \end{pmatrix}.$$

很明显，群 $G$ 是在动作集合 $\{a,b,c,d\}$ 上的不变群. 事件集合 $E_{\mathcal{E}_P}$ 中的每个事件的轨道是 $\theta(e_1)=\theta(e_2)=\{e_1,e_2\}$，$\theta(e_3)=\theta(e_4)=\{e_3,e_4\}$，$\theta(e_5)=\{e_5\}$ 和 $\theta(e_6)=\{e_6\}$. 因此，事件结构 $\mathcal{E}_P$ 的商事件结构 $\mathcal{E}_{PG}=\langle E_G,<_G,\#_G,l_G\rangle$:

$E_G=\{\ \{e_1,e_2\},\{e_3,e_4\},\{e_5\},\{e_6\}\ \}$,

$<_G=\{\ (\{e_5\},\{e_6\})\ \}$,

$\#_G=\{\ (\{e_1,e_2\},\{e_5\}),(\{e_5\},\{e_1,e_2\}),(\{e_5\},\{e_3,e_4\}),(\{e_3,e_4\},\{e_5\}),(\{e_1,e_2\},\{e_6\}),(\{e_6\},\{e_1,e_2\}),(\{e_3,e_4\},\{e_6\}),(\{e_6\},\{e_3,e_4\})\ \}$,

$l_G(\{e_1,e_2\})=a, l_G(\{e_3,e_4\})=b, l_G(\{e_5\})=c, l_G(\{e_6\})=d.$

显然，商事件结构 $\mathcal{E}_{PG}$ 对应进程 $(a||b)+(c;d)$.

上面给出了事件结构的商模型的概念，接着研究商模型与原事件结构存在哪些关系. 下面的命题表明：事件结构的商模型仍是事件结构，它们的动作集是相同的并且它们的格局存在特殊关系.

**命题 4.2.1** 设事件结构 $\mathcal{E}=\langle E,<,\#,l\rangle\in E$(用 $\mathrm{Act}_{\mathcal{E}}(\mathrm{Act}_{\mathcal{E}}\subseteq\mathrm{Act})$ 表示 $\varepsilon$ 涉及的动作的集合), 设 $G$ 是动作集合 $\mathrm{Act}_{\mathcal{E}}$ 中的不变群, 并设 $\mathcal{E}_G=\langle E_G,<_G,\#_G,l_G\rangle$ 是 $\mathcal{E}$ 的商事件结构, 以及 $\mathrm{Act}_{\mathcal{E}_G}$ 是 $\mathcal{E}_G$ 的动作集合, 那么,

(1) $\mathrm{Act}_{\mathcal{E}_G}=\mathrm{Act}_{\mathcal{E}}$；

(2) $\forall e_1,e_2\in E$: $e_1<e_2\Rightarrow\theta(e_1)<_G\theta(e_2)$, $e_1>e_2\Rightarrow\theta(e_1)>_G\theta(e_2)$ 和 $e_1\ \mathrm{co}\ e_2\Rightarrow\theta(e_1)\ \mathrm{co}_G\ \theta(e_2)$；

(3) $\#_G$ 是对称的，并且 $\mathcal{E}_G$ 是事件结构；

(4) $\forall e_1,e_2\in E$: $e_1\#e_2\not\Rightarrow\theta(e_1)\#_G\theta(e_2)$, 但 $\theta(e_1)\#_G\theta(e_2)\Rightarrow\forall d_1\in\theta(e_1),\forall d_2\in\theta(e_2):d_1\#d_2$；

(5) $\forall X:X\in C(\mathcal{E})\ \Rightarrow\ \exists Y:Y\in C(\mathcal{E}_G)\wedge X\simeq_p Y$，并且 $\forall Y:Y\in C(\mathcal{E}_G)\ \Rightarrow\ \exists X:X\in C(\mathcal{E})\wedge Y\simeq_p X$.

**证明** (1) 和 (2) 的证明很容易获得，此处略.

(3) 先证明 $\#_G$ 是对称的. 这里用反证法. 假设 $\#_G$ 不是对称的，那么 $\exists e_1,e_2\in E:\theta(e_1)\#_G\theta(e_2)\not\Rightarrow\theta(e_2)\#_G\theta(e_1)$，即 $\theta(e_2)<_G\theta(e_1)$ 或 $\theta(e_2)>_G\theta(e_1)$ 或 $\theta(e_2)\ \mathrm{co}_G\ \theta(e_1)$. 因为 co 是对称的，所以 $\mathrm{co}_G$ 也是对称的. 因此，仅仅存在 $\theta(e_2)<_G\theta(e_1)$ 或 $\theta(e_2)>_G\theta(e_1)$. 而且，由于 $<_G$ 的逆是 $>_G$，则 $\theta(e_2)<_G\theta(e_1)\Rightarrow\theta(e_1)>_G\theta(e_2)$ 或 $\theta(e_2)>_G\theta(e_1)\Rightarrow\theta(e_1)<_G\theta(e_2)$，这与 $\theta(e_1)\#_G\theta(e_2)$ 矛盾. 因此，$\#_G$ 是对称的. 接

下来，证明 $\mathcal{E}_G$ 是一个事件结构. 根据定义 4.2.3 和定义 2.2.1，并且因为 $\#_G$ 是对称的，我们直接有这个结论.

(4) 根据定义 4.2.3，商事件结构保留了原事件结构的原因关系和独立关系，但矛盾关系可能被约简. 因此，有 $\forall e_1, e_2 \in E$: $e_1 \# e_2 \not\Rightarrow \theta(e_1)\#_G\theta(e_2)$. 接下来证明 $\theta(e_1)\#_G\theta(e_2) \Rightarrow \forall d_1 \in \theta(e_1), \forall d_2 \in \theta(e_2) : d_1 \# d_2$. 用反证法. 假设 $\theta(e_1)\#_G\theta(e_2) \not\Rightarrow \forall d_1 \in \theta(e_1), \forall d_2 \in \theta(e_2) : d_1 \# d_2$，那么 $\theta(e_1)\#_G\theta(e_2) \Rightarrow \exists e_1' \in \theta(e_1), \exists e_2' \in \theta(e_2) : (e_1' < e_2') \vee (e_1' > e_2') \vee (e_1' \text{ co } e_2')$. 又根据定义 4.2.3，有 $\theta(e_1') <_G \theta(e_2')$ 或 $\theta(e_1') >_G \theta(e_2')$ 或者 $\theta(e_1') \text{ co}_G\ \theta(e_2')$，这些都与 $\theta(e_1)\#_G\theta(e_2)$(因为 $\theta(e_1') = \theta(e_1)$, $\theta(e_2') = \theta(e_2)$) 矛盾. 所以，上述结论成立.

(5) 先证明 $\forall X : X \in C(\mathcal{E}) \ \Rightarrow \ \exists Y : Y \in C(\mathcal{E}_G)$. 因为 $G$ 是动作集合 $\text{Act}_{\mathcal{E}}$ 所有动作的不变群，因而能构造集合 $Y = \{\theta(e) \mid e \in X,\ l_G(\theta(e)) = l(e)\}$. 假设 $Y$ 中没有矛盾关系，根据定义 2.2.2，有 $\exists(\theta(e_1),\ \theta(e_2)) \in \#_G$，这里 $\theta(e_1),\ \theta(e_2) \in Y$. 所以根据上面 (4) 的结论，有 $(e_1,\ e_2) \in \#$，这与集合 $X$ 中不存在矛盾关系相矛盾. 因此 $Y$ 中没有矛盾关系. 另一方面，由于 $X$ 是左封闭的，对所有 $d,\ e \in E$: $e \in X \wedge d < e \Rightarrow d \in X$. 显然，存在对所有 $\theta(d),\ \theta(e) \in E_G$: $\theta(e) \in X \wedge \theta(d) <_G \theta(e) \Rightarrow \theta(d) \in Y$，所以 $Y$ 是左封闭的. 很明显，根据定义 4.2.3，则有 $Y \subseteq E_G$，因而有 $Y \in C(\mathcal{E}_G)$.

假设 $|X| \neq |Y|$. 不失一般性，设 $|X| > |Y|$，则 $\exists e, d \in X : e \neq d \wedge \theta(d) = \theta(e)$. 因为 $e, d \in X \wedge e \neq d$，所以有 $(e, d) \in < \cup > \cup \text{ co}$. 又根据定义 4.2.3，则有 $(\theta(e), \theta(d)) \in <_G \cup >_G \cup \text{ co}_G$，这与 $\theta(d) = \theta(e)$ 矛盾. 所以 $|X| = |Y|$.

显然，根据定义 4.2.3，有 $e \in X \Leftrightarrow \theta(e) \in Y \wedge l(e) = l(\text{rep}(\theta(e))) = l_G(\theta(e))$ 和 $e, d \in X \wedge e < d \Leftrightarrow \theta(e), \theta(d) \in Y \wedge \theta(e) <_G \theta(d)$，即 $X$ 和 $Y$ 之间存在一个双射，所以 $X \simeq_p Y$.

接下来，证明 $\forall Y : Y \in C(\mathcal{E}_G) \ \Rightarrow \ \exists X : X \in C(\mathcal{E}) \wedge Y \simeq_{\text{p}} X$. 构造一个集合 $X$，使得对所有 $e, d \in X$: $\theta(e), \theta(d) \in Y \wedge \theta(e) \neq \theta(d)$，$\theta(e) <_G \theta(d) \Rightarrow e < d$ 和 $\theta(e) >_G \theta(d) \Rightarrow e > d$，$\theta(e) \text{ co}_G\ \theta(d) \Rightarrow e \text{ co } d$，$|X| = |Y|$，也就是说，从集合 $Y$ 中的每个元素 (轨道) 中选出一个事件，使得它们在集合 $X$ 中的元素间满足原因和独立关系. 显然，根据定义 4.2.3，能构造出集合 $X$. 现在，证明 $X \in C(\mathcal{E})$. 因为 $Y$ 是左封闭的并且没有矛盾关系，则 $X$ 是左封闭的并且没有矛盾关系. 因而 $X \in C(\mathcal{E})$. 很明显，对所有 $e, d \in X$: $\theta(e) <_G \theta(d) \Rightarrow e < d$，$|X| = |Y|$. 并且根据定义 4.2.3，则有 $e < d \Rightarrow \theta(e) <_G \theta(d)$ 和 $l(e) = l_G(\theta(e))$. 所以在 $X$ 和 $Y$ 之间存在一个双射，即 $X \simeq_{\text{p}} Y$.

命题 4.2.1(2) 表明商模型保留了原事件结构的原因和独立关系，但命题 4.2.1(4) 却表明了矛盾关系被约简. 注意在一个轨道中的任何两个不同的事件实际上是互相矛盾的. 同时命题 4.2.1(5) 也表明商模型保留了原事件结构的所有行为. 此外，根据该命题，直接有下面的推论.

**推论 4.2.1** 设事件结构 $\mathcal{E} \in \boldsymbol{E}$( $\mathrm{Act}_{\mathcal{E}}(\mathrm{Act}_{\mathcal{E}} \subseteq \mathrm{Act})$ 是动作集), 设 $G$ 是动作集合 $\mathrm{Act}_{\mathcal{E}}$ 中的不变群, 并设 $\mathcal{E}_G = \langle E_G, <_G, \#_G, l_G \rangle$ 是 $\mathcal{E}$ 的商事件结构. 那么 $X = \{e_1, \cdots, e_n\} \in C(\mathcal{E}) \Rightarrow Y = \{\theta(e_1), \cdots, \theta(e_n)\} \in C(\mathcal{E}_G) \wedge X \simeq_{\mathrm{e}} Y$, 这里 $\theta(e_i)$ 是 $e_i$ 的轨道 $(i = 1, \cdots, n)$.

根据推论 4.2.1, 如果 $X = \{e_1, \cdots, e_n\} \in C(\mathcal{E})$, 那么存在 $Y = \{\theta(e_1), \cdots, \theta(e_n)\} \in C(\mathcal{E}_G)$, 其中 $\theta(e_i)$ 是 $e_i (i = 1, \cdots, n)$ 的轨道. 格局 $Y$ 称作 $X$ 的 **商格局**. 所以, 上面的命题也蕴含下面的推论.

**推论 4.2.2** 设事件结构 $\mathcal{E} \in \boldsymbol{E}$( $\mathrm{Act}_{\mathcal{E}}(\mathrm{Act}_{\mathcal{E}} \subseteq \mathrm{Act})$ 表示 $\Sigma$ 涉及的动作集), 设 $G$ 是动作集合 $\mathrm{Act}_{\mathcal{E}}$ 中的不变群, 并设 $\mathcal{E}_G$ 是 $\mathcal{E}$ 的商事件结构. 如果 $X, X' \in C(\mathcal{E})(X \neq X')$, 并且 $X, X'$ 对应相同的商格局 $Y \in C(\mathcal{E}_G)$, 则 $X \simeq_{\mathrm{p}} X' \simeq_{\mathrm{p}} Y$.

注意, 为了方便, 本书中一个事件结构的商事件结构, 并有一个在其动作集合上的不变群, 被简称为“该事件结构的商事件结构”.

## 4.3 对称与等价

在这部分, 先引入交织迹等价 (interleaving trace equivalence) 和交织互模拟等价 (interleaving bisimulation equivalence)[41], 这两类等价与 3.4 节的定义本质上是一致的, 但这里的等价是建立在事件结构模型上的, 又存在一些差别. 我们先讨论事件结构上的单个动作变迁; 接着讨论商事件结构和原事件结构是否交织迹和交织互模拟等价; 最后引入基于偏序执行的等价 —— 偏序多集迹等价 (pomset trace equivalence).

**定义 4.3.1**(单个动作变迁) 设事件结构 $\mathcal{E} \in E$. 一个变迁 $X \xrightarrow{a}_{\mathcal{E}} X'$ 称作 **单个动作变迁**当且仅当 $a \in \mathrm{Act}, X, X' \in C(\mathcal{E}), X \subseteq X'$, 并且 $\exists e \in E_{\mathcal{E}}$: $X' - X = e(l_{\mathcal{E}}(e) = a)$.

这里, $X \xrightarrow{a}_{\mathcal{E}} X'$ 表示在事件结构 $\mathcal{E}$ 中, 通过执行动作 $a$, 格局 $X$ 表示的状态可能进化成 $X'$ 表示的状态. 这个变迁关系关联着一个标记变迁系统和一个事件结构. 因此, 定义在标记变迁系统上的等价直接诱导出定义在事件结构上对应的等价.

根据定义 2.4.1, 给定一个事件结构 $\mathcal{E} = \langle E, <, \#, l \rangle \in E$, 事实上, 它的行为由其诱导出的变迁系统 $\langle C(\mathcal{E}), \mathrm{Act}_{\mathcal{E}}, \rightarrow, \emptyset \rangle$ 展示, 其中所有格局 $C(\mathcal{E})$ 是状态集合, 标记集合是动作集合 $\mathrm{Act}_{\mathcal{E}}$, 变迁 $\rightarrow$ 是 $C(\mathcal{E})$ 中每两个格局之间的单个动作变迁, 以及初始格局空集 $\emptyset$ 是初始状态.

**例 4.3.1** 这里, 继续考虑例 4.2.1. 图 4.1 和图 4.2 分别描述了事件结构 $\mathcal{E}_P$ 和商事件结构 $\mathcal{E}_{PG}$ 的单个动作变迁.

上面引入的单个动作变迁, 使得我们能够讨论商事件结构和原事件结构的单

个动作变迁之间存在的关系. 下面的命题将说明这一点.

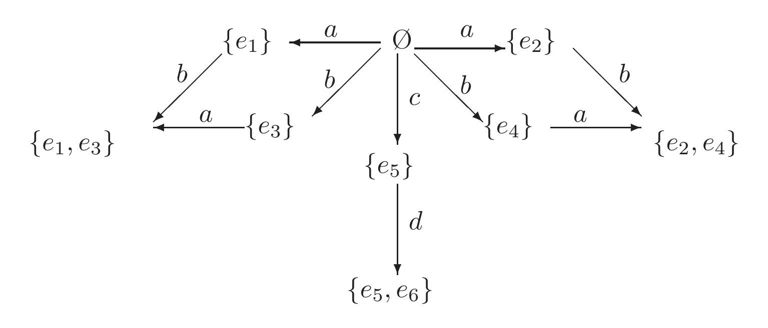



图 4.1 事件结构 $\mathcal{E}_P$ 的变迁和格局

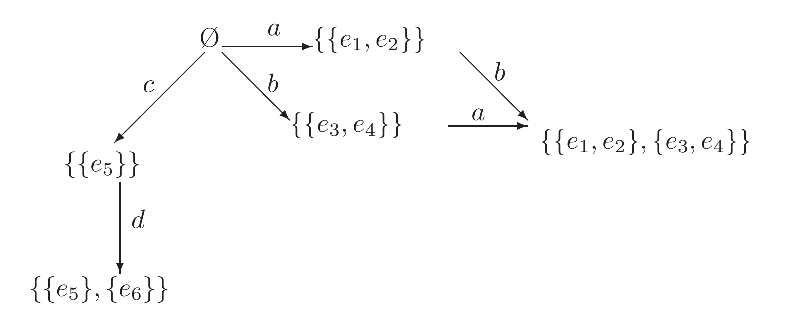



图 4.2 事件结构 $\mathcal{E}_{PG}$ 的变迁和格局

**命题 4.3.1** 设事件结构 $\mathcal{E} = \langle E, <, \#, l\rangle \in E$, 并设 $\mathcal{E}_G = \langle E_G, <_G, \#_G, l_G\rangle$ 是 $\mathcal{E}$ 的商事件结构, 则有

(1) $\forall X, X' \in C(\mathcal{E}), \forall a \in \text{Act}$: $X \xrightarrow{a}_{\mathcal{E}} X' \Rightarrow \exists Y, Y' \in C(\mathcal{E}_G)$: $Y \xrightarrow{a}_{\mathcal{E}_G} Y'$;

(2) $\forall Y, Y' \in C(\mathcal{E}_G), \forall a \in \text{Act}$: $Y \xrightarrow{a}_{\mathcal{E}_G} Y' \Rightarrow \exists X, X' \in C(\mathcal{E})$: $X \xrightarrow{a}_{\mathcal{E}} X'$.

**证明** (1) 先证明 $\forall X, X' \in C(\mathcal{E}), \forall a \in \text{Act}: X \xrightarrow{a}_{\mathcal{E}} X' \Rightarrow \exists Y, Y' \in C(\mathcal{E}_G): Y \xrightarrow{a}_{\mathcal{E}_G} Y'$. 因为 $\mathcal{E}_G$ 是事件结构 $\mathcal{E}$ 的商事件结构, 并且存在一个不变群 $G$, 而且因为 $X \xrightarrow{a}_{\mathcal{E}} X', a \in \text{Act}_{\mathcal{E}}$ 和命题 4.2.1, 能构造集合 $Y = \{\theta(e) \mid e \in X, l_G(\theta(e)) = l(e)\}$ 和 $Y' = \{\theta(e') \mid e' \in X', l_G(\theta(e')) = l(e')\}$, 并使得 $Y, Y' \in C(\mathcal{E}_G)$: $X \simeq_{\text{p}} Y$ 和 $X' \simeq_{\text{p}} Y'$. 根据定义 4.3.1, 假定 $X' - X = \{d\}$. 因此 $X \subset X', l(d) = a, d \notin X, d \in X'$, 并且, 若令 $|X|$ 和 $|X'|$ 分别表示 $X$ 和 $X'$ 的元素个数, 则 $|X'| - |X| = 1$. 显然, 直接有 $Y \subset Y', |Y'| - |Y| = 1$ 和 $Y' - Y = \{\theta(d)\}$. 又因为动作 $a$ 是 $G$ 的一个不变量, 则有 $l(\text{rep}(\theta(d))) = l_G(\theta(d)) = a$. 所以, 有单个动作变迁 $Y \xrightarrow{a}_{\mathcal{E}_G} Y'$.

(2) 为了证明 $\forall Y, Y' \in C(\mathcal{E}_G), \forall a \in \text{Act} : Y \xrightarrow{a}_{\mathcal{E}_G} Y' \Rightarrow \exists X, X' \in C(\mathcal{E}) : X \xrightarrow{a}_{\mathcal{E}} X'$, 用与证明命题 4.2.1(5) 类似的方法构造集合 $X$ 和 $X'$ , 并且根据上面

(1) 的证明，这个结论很容易得出.

接下来将研究另外一些重要的等价概念. 先讨论交织等价. 为了定义交织迹等价，需要先给出“迹”的定义.

**定义 4.3.2**(迹) 设事件结构 $\mathcal{E} \in E$. 一个字 $w = a_1 \cdots a_n \in \mathrm{Act}^*$ 是 $\mathcal{E}$ 的一个**迹**，当且仅当 $\exists X_0, \cdots, X_n \in C(\mathcal{E}) : X_0 = \emptyset$ 和 $X_{i-1} \xrightarrow{a_i} X_i, i = 1, \cdots, n$.

这里，$\mathrm{tr}s(\mathcal{E})$ 表示事件结构 $\mathcal{E}$ 的所有迹的集合. 接下来定义交织迹等价.

**定义 4.3.3** 设两事件结构 $\mathcal{E}, \mathcal{F} \in E$. $\mathcal{E}$ 和 $\mathcal{F}$ 称作**交织迹等价**(表示为 $\mathcal{E} \approx_{\mathrm{it}} \mathcal{F}$ )，当且仅当 $\mathrm{tr}s(\mathcal{E}) = \mathrm{tr}s(\mathcal{F})$.

有了交织迹等价概念，就能够考察商事件结构和原事件结构是否交织迹等价的.

**定理 4.3.1** 设事件结构 $\mathcal{E} \in E$，并设 $\mathcal{E}_G$ 是 $\mathcal{E}$ 的商事件结构，则 $\mathcal{E}_G \approx_{\mathrm{it}} \mathcal{E}$.

**证明** 用反证法. 假设 $\mathcal{E}_G \not\approx_{\mathrm{it}} \mathcal{E}$. 如果 $\mathrm{tr}s(\mathcal{E}), \mathrm{tr}s(\mathcal{E}_G)$ 分别表示 $\mathcal{E}, \mathcal{E}_G$ 的所有迹的集合，则 $\mathrm{tr}s(\mathcal{E}) \neq \mathrm{tr}s(\mathcal{F})$. 因此，不失一般性，假定迹 $\omega = a_1 \cdots a_n \in \mathrm{tr}s(\mathcal{E})$，并且 $\omega \notin \mathrm{tr}s(\mathcal{E}_G)$. 根据定义 4.3.2，则有格局 $X_0, \cdots, X_n \in C(\mathcal{E}) : X_0 = \emptyset$ ，并且有 $X_{i-1} \xrightarrow{a_i} X_i, i = 1, \cdots, n$. 又因为命题 4.3.1，所以有 $Y_0, \cdots, Y_n \in C(\mathcal{E})$ ，并且 $Y_i \simeq_{\mathrm{p}} X_i, i = 1, \cdots, n$ 和 $Y_{i-1} \xrightarrow{a_i} Y_i, i = 1, \cdots, n$. 显然，根据定义 4.3.2，有 $\omega \in \mathrm{tr}s(\mathcal{E}_G)$，与 $\omega \notin \mathrm{tr}s(\mathcal{E}_G)$ 矛盾. 所以 $\mathcal{E}_G \approx_{\mathrm{it}} \mathcal{E}$ 成立.

接下来引入交织互模拟等价的定义 [41]. 注意，有人可能直接称其为“互模拟等价”，没有“交织”这两个字. 但根据文献 [41]，称“交织互模拟等价”更准确.

**定义 4.3.4**(交织互模拟等价) 设两事件结构 $\mathcal{E}, \mathcal{F} \in E$. 一个关系 $R \subseteq C(\mathcal{E}) \times C(\mathcal{F})$ 称作 $\mathcal{E}$ 和 $\mathcal{F}$ 之间的一个 **交织互模拟**，当且仅当 $(\emptyset, \emptyset) \in R$，并且如果 $(X, Y) \in R$，那么

- $X \xrightarrow{a}_{\mathcal{E}} X', a \in \mathrm{Act} \Rightarrow \exists Y' : Y \xrightarrow{a}_{\mathcal{F}} Y' \wedge (X', Y') \in R$;
- $Y \xrightarrow{a}_{\mathcal{F}} Y', a \in \mathrm{Act} \Rightarrow \exists X' : X \xrightarrow{a}_{\mathcal{E}} X' \wedge (X', Y') \in R$.

$\mathcal{E}$ 与 $\mathcal{F}$ 称作 **交织互模拟等价** (表示为 $\mathcal{E} \approx_{\mathrm{ib}} \mathcal{F}$ )，当且仅当 $\mathcal{E}$ 和 $\mathcal{F}$ 之间存在一个交织互模拟.

现在考察商事件结构和原事件结构是否交织互模拟等价.

**定理 4.3.2** 设事件结构 $\mathcal{E} \in E$，并设 $\mathcal{E}_G$ 是 $\mathcal{E}$ 的商事件结构，则 $\mathcal{E}_G \approx_{\mathrm{ib}} \mathcal{E}$.

**证明** 设 $C(\mathcal{E}), C(\mathcal{E}_G)$ 分别是 $\mathcal{E}, \mathcal{E}_G$ 的所有格局的集合. 因为命题 4.3.1，则有 $X \xrightarrow{a}_{\mathcal{E}} X', a \in \mathrm{Act} \Rightarrow \exists Y' : Y \xrightarrow{a}_{\mathcal{E}_G} Y'$ 和 $Y \xrightarrow{a}_{\mathcal{F}} Y', a \in \mathrm{Act} \Rightarrow \exists X' : X \xrightarrow{a}_{\mathcal{E}} X'$. 构造一个集合 $R'$：(1) 令 $(\emptyset, \emptyset)$ 属于该集合 $R'$; (2) 令 $(X, Y)$ 属于 $R'$，并且令 $(X', Y')$ 也属于 $R'$; (3) 如果 $X', Y'$ 通过单个动作变迁分别存在后继格局，则令所有配对 ($X'$ 可达的格局与 $Y'$ 对应到达的格局) 都属于集合 $R'$. 显然，$R' \subseteq C(\mathcal{E}) \times C(\mathcal{E}_G)$，并且 $R'$ 事实上是 $\mathcal{E}$ 和 $\mathcal{E}_G$ 的一个交织互模拟关系. 所以 $\mathcal{E}_G \approx_{\mathrm{ib}} \mathcal{E}$ 成立.

上述定理表明商事件结构和原事件结构是交织互模拟等价的. 很显然，根据定理 4.3.1 和定理 4.3.2，直接获得下面的推论.

**推论 4.3.1** 设事件结构 $\mathcal{E} \in E$，并设 $\mathcal{E}_G$ 是 $\mathcal{E}$ 的商事件结构. 如果变迁系统 $\mathrm{LTS} = \langle C(\mathcal{E}), \mathrm{Act}, \to_{\mathcal{E}}, \emptyset\rangle$ 和 $\mathrm{LTS}_G = \langle C(\mathcal{E}_G), \mathrm{Act}, \to_{\mathcal{E}_G}, \emptyset\rangle$ 分别是 $\mathcal{E}$ 和 $\mathcal{E}_G$ 诱导得出的，则 $\mathrm{LTS} \approx_{\mathrm{it}} \mathrm{LTS}_G$ ，并且 $\mathrm{LTS} \approx_{\mathrm{ib}} \mathrm{LTS}_G$.

事实上，交织迹等价和交织互模拟等价都是基于交织模型，即基于单个动作变迁的标记变迁系统. 这两类等价是两类重要的交织等价，研究它们具有重要的意义. 最后，引入另外一类等价，它是基于偏序执行建立起来的，称作“偏序多集迹等价 (pomset trace equivalence)”[41]. 同样，我们还将研究商事件结构和原事件结构是否偏序多集迹等价.

**定义 4.3.5**(偏序多集迹等价) 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$，并设 $\mathrm{pomsets}(\mathcal{E}) = \{\mathrm{pomset}(X) \mid X \in C(\mathcal{E})\}$ 和 $\mathrm{pomsets}(\mathcal{F}) = \{\mathrm{pomset}(Y) \mid Y \in C(\mathcal{F})\}$. $\mathcal{E}$ 和 $\mathcal{F}$ 称作**偏序多集迹等价**(表示为 $\mathcal{E} \approx_{\mathrm{pt}} \mathcal{F}$)，当且仅当 $\mathrm{pomsets}(\mathcal{E}) = \mathrm{pomsets}(\mathcal{F})$.

下面的定理表明商事件结构和原事件结构是偏序多集迹等价的.

**定理 4.3.3** 设事件结构 $\mathcal{E} \in E$，并设 $\mathcal{E}_G$ 是 $\mathcal{E}$ 的商事件结构，则 $\mathcal{E}_G \approx_{\mathrm{pt}} \mathcal{E}$.

**证明** 用反证法. 假设 $\mathcal{E}_G \not\approx_{\mathrm{pt}} \mathcal{E}$，则有 $\mathrm{pomsets}(\mathcal{E}) \neq \mathrm{pomsets}(\mathcal{E}_G)$. 因此，不失一般性，设 $X \in \mathrm{pomsets}(\mathcal{E})$ ，并且 $X \notin \mathrm{pomsets}(\mathcal{E}_G)$. 显然，$X \in C(\mathcal{E})$ 并且 $\nexists Y \in C(\mathcal{E}_G) : X \simeq_{\mathrm{p}} Y$，这与命题 4.2.1( 5 ) 矛盾. 所以 $\mathcal{E}_G \approx_{\mathrm{pt}} \mathcal{E}$ 成立.

本节已经证明商事件结构交织迹、交织互模拟和偏序多集迹等价于原事件结构，这充分说明商事件结构保留了原事件结构的行为和原因偏序关系. 下一节将进一步揭示它们之间的关系.

## 4.4 动作细化的保持

这部分将讨论商事件结构和原事件结构之间的交织迹等价、交织互模拟等价和偏序多集迹等价在动作细化下的保持问题. 这里，先建立商事件结构和原事件结构之间的行为关系.

**命题 4.4.1** 设事件结构 $\mathcal{E} \in E$，设 $\mathcal{E}_G$ 是 $\mathcal{E}$ 的商事件结构，并设 ref 是一个细化函数. 如果对任意的 $X \in C(\mathcal{E})$，存在格局 $X$ 的商格局 $Y \in C(\mathcal{E}_G)$，则 $\mathrm{ref}(X) \simeq_{\mathrm{p}} \mathrm{ref}(Y)$.

**证明** 因为存在事件结构 $\mathcal{E}$ 动作集合中的所有动作的一个不变群，所以有 $\forall e \in X \Rightarrow l_{\mathcal{E}}(e) = l_{\mathcal{E}_G}(\theta(e))$(其中 $\theta(e)$ 是事件 $e$ 的轨道)，那么有 $\mathrm{ref}(l_{\mathcal{E}}(e)) \simeq_e \mathrm{ref}(l_{\mathcal{E}_G}(\theta(e)))$. 显然，如果 $W, V$ 是两个格局，$d, d'$ 是两个事件并且 $W \simeq_{\mathrm{p}} V$，则有 $\{d\} \times W \simeq_{\mathrm{p}} \{d'\} \times V$. 因此，如果 $\theta(e)$ 是事件 $e \in X(X \in C(\mathcal{E}))$ 的轨道，并且 $X_e \in C(\mathrm{ref}(l_{\mathcal{E}}(e))) - \{\emptyset\}$，$Y_{\theta(e)} \in C(\mathrm{ref}(l_{\mathcal{E}_G}(e))) - \{\emptyset\}$，$X_e \simeq_{\mathrm{p}} Y_{\theta(e)}$，则

$\{e\} \times X_e \simeq_{\mathrm{p}} \{\theta(e)\} \times Y_{\theta(e)}$. 确实存在这种情况 (因为 $\mathrm{ref}(l_{\mathcal{E}}(\mathrm{e})) \simeq_{\mathrm{e}} \mathrm{ref}(\mathrm{l}_{\mathcal{E}_{\mathrm{G}}}(\theta(\mathrm{e})))$). 假设两个格局 $X$ 和 $Y$ 分别被函数 ref 细化，则有 $\widetilde{X} = \bigcup\limits_{e \in X} \{e\} \times X_e$, 其中 $e \in X : X_e \in C(\mathrm{ref}(l_{\mathcal{E}}(e))) - \{\emptyset\}$; $\widetilde{Y} = \bigcup\limits_{\theta(e) \in Y} \{\theta(e)\} \times Y_{\theta(e)}$ 其中 $\theta(e) \in Y : Y_{\theta(e)} \in C(\mathrm{ref}(l_{\mathcal{E}_G}(e))) - \{\emptyset\}$. 而且因为推论 4.2.2，所以有 $X \simeq_{\mathrm{p}} Y$. 因此 $\widetilde{X} \simeq_{\mathrm{p}} \widetilde{Y}$ 成立. 显然，因为 $X \simeq_{\mathrm{p}} Y$，如果 $e \in \mathrm{busy}(\widetilde{X}) \Rightarrow e$ 是集合 $X$ 中关于 $<_{\mathcal{E}}$ 最大的，其中 $\mathrm{busy}(\widetilde{X}) := \{e \in X \mid X_e$ 不是终止格局 $\}$，则 $\theta(e) \in \mathrm{busy}(\widetilde{Y}) \Rightarrow \theta(e)$ 是集合 $Y$ 中关于 $<_G$ 最大的，其中 $\mathrm{busy}(\widetilde{Y}) := \{\theta(e) \in Y \mid Y_e$ 不是终止格局 $\}$. 所以根据定义 2.3.2, $\mathrm{ref}(X) \simeq_{\mathrm{p}} \mathrm{ref}(Y)$ 成立.

上述命题表明, 如果任何格局 $X$ 和它的商格局 $Y$ 被相同的细化函数细化，细化后得到的格局是同构的，即

$$\mathrm{ref}(\mathcal{E}|_X) \simeq_{\mathrm{e}} \mathrm{ref}(\mathcal{E}_G|_Y).$$

因此，商事件结构和原事件结构有相同的行为. 接下来，讨论各种等价在动作细化下的保持问题.

**定理 4.4.1** 设事件结构 $\mathcal{E} \in E$, 设 $\mathcal{E}_G$ 是 $\mathcal{E}$ 的商事件结构，并设 ref 是一个细化函数，则有

(1) $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{pt}} \mathrm{ref}(\mathcal{E}_G)$,

(2) $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{it}} \mathrm{ref}(\mathcal{E}_G)$,

(3) $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{E}_G)$.

**证明** (1) 根据定理 4.3.3 和文献 [41] 中的定理 8.1，直接获得结论 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{pt}} \mathrm{ref}(\mathcal{E}_G)$.

(2) 根据上面 (1) 的结论：$\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{pt}} \mathrm{ref}(\mathcal{E}_G)$，以及由于事件结构同构有相同的动作标签，所以直接有 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{it}} \mathrm{ref}(\mathcal{E}_G)$.

(3) 根据命题 4.3.1(1) 可得：$\forall X, X' \in C(\mathcal{E}), \forall a \in \mathrm{Act}:\ X \xrightarrow{a}_{\mathcal{E}} X' \Rightarrow \exists Y, Y' \in C(\mathcal{E}_G) : Y \xrightarrow{a}_{\mathcal{E}_G} Y'$；并且根据命题 4.2.1(5) 和推论 4.2.1 可知,$Y, Y'$ 分别是 $X, X'$ 的商格局. 而且，因为命题 2.3.2，所以 $\mathrm{ref}(X) \simeq_{\mathrm{p}} \mathrm{ref}(Y)$ 和 $\mathrm{ref}(X') \simeq_{\mathrm{p}} \mathrm{ref}(Y')$. 也就是说，$\mathrm{ref}(\mathcal{E}|_X) \simeq_{\mathrm{e}} \mathrm{ref}(\mathcal{E}_G|_Y)$ 和 $\mathrm{ref}(\mathcal{E}|_{X'}) \simeq_{\mathrm{e}} \mathrm{ref}(\mathcal{E}_G|_{Y'})$. 因此，有 $\mathrm{ref}(\mathcal{E}|_{X'}) \approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{E}_G|_{Y'})$, 即存在一个关系 $R \subseteq C(\mathrm{ref}(\mathcal{E}|_{X'})) \times C(\mathrm{ref}(\mathcal{E}|_{Y'}))$ 是一个交织互模拟. 又因为 $X', Y'$ 可能是终止格局，所以 $R$ 是 $\mathrm{ref}(\mathcal{E})$ 和 $\mathrm{ref}(\mathcal{E}_G)$ 之间的一个交织互模拟. 因此，结论 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{E}_G)$ 成立.

文献 [41] 阐明了, 在一般情况下, 交织迹等价和交织互模拟等价在动作细化下不保持. 但是，在商事件结构和原事件结构之间的这两种交织等价在动作细化下是保持的，当然偏序多集迹等价在动作细化下也是保持的. 这些性质表明在事件结构中对称约简对可能需要验证分析的性质没有影响.

## 4.5 对称约简算法

本节建立针对事件结构的对称约简算法，同时给出实例验证算法的正确性.

为了给出对称约简算法，必须仔细考虑事件结构本身的性质. 上一章已经说明，一个事件结构存在对称性意味着它存在自同构，并且商事件结构就是原事件结构约简后的结构. 定义 4.2.3 和命题 4.2.1(2) 表明事件结构中的矛盾关系被约简，同构的原因关系和独立关系可能被保留一份. 很自然地想到，按一定要求把一个事件结构分解成一些子结构，特别是同构的部分分别属于不同的子结构，如果只保留同构的子结构中的一份，这样得到的事件结构与定义 4.2.3 所定义的商事件结构同构. 根据定义 2.2.1，在事件结构中，矛盾继承规则和任何事件都有唯一的前驱，可以将事件结构分解成一些 **相互矛盾的子结构**(我们这样称谓)：一个子结构的任何一个事件与另一个子结构的每个事件彼此相互矛盾. 因此，对称约简算法主要包含以下三个步骤：

(1) 将事件结构分解成相互矛盾的子结构；

(2) 判断是否存在同构的子结构；

(3) 把同构的子结构只保留一份，并合并其他的子结构，即得约简后的事件结构.

上述第一步很容易实现，但第二步较难. 为了叙述方便，先给出几个定义.

设 $A$ 是一个集合，$|A|$ 表示集合 $A$ 中元素的个数.

**定义 4.5.1**(子结构) 设 $\mathcal{E} \in \boldsymbol{E}$ 和 $\mathcal{E} \neq \varnothing$. 一个结构 $S = \langle E^s, <^s, \#^s, l^s \rangle$ 是事件结构 $\mathcal{E}$ 的一个 **子结构**，当且仅当 $\exists E^s \subseteq E_{\mathcal{E}}$: $E^s \neq \emptyset$, $E_{\mathcal{E}} \setminus E^s \neq \emptyset \Rightarrow \forall e \in E^s, \forall e' \in E_{\mathcal{E}} \setminus E^s : e \# e'$，并且 $S = \mathcal{E}|_{E^s}$.

显然，一个事件结构的任何子结构也是事件结构. 将一个事件结构分解成子结构的算法在算法 4.5.2 中给出描述，其中的分解过程明显是正确的，并且能在有限步完成. 比如，进程 $(a||b) + (c;d) + f$ 对应事件结构 $\mathcal{E}$，算法能将事件结构 $\mathcal{E}$ 分解成三个子结构：$S_1$，$S_2$ 和 $S_3$，它们分别对应子进程 $a||b$, $c;d$ 和 $f$.

接下来研究怎样计算两个事件结构同构的问题. 同样为了方便，先给出一个定义. 设集合 $E$ 是被动作集合 Act 标记的事件集合，称动作集合 $L(E) = \{a \in \mathrm{Act} \mid \exists e \in E : l(e) = a\}$ 是 $E$ 的 **动作标记集合**.

因为两个事件结构的同构要求它们之间存在一个双射，事件保持原因、矛盾和标记关系. 因此，同构计算的算法包含以下三步：

(1) 判断两个事件结构的事件集合之间是否存在一个双射，并判断它们的动作标记集合是否一致 (相同)；

(2) 计算两个事件结构的事件之间的原因关系在相同的动作标记下是否保持相同；

(3) 计算两个事件结构的事件之间的矛盾关系在相同的动作标记下是否保持相同.

以上任何一步不成立，该算法立即终止，说明两个事件结构不同构；如果全部通过，即说明它们是同构的. 下面给出算法.

**算法 4.5.1** computing-isomorphism($S_1, S_2$)(计算两个事件结构是否同构)

输入： $S_1 = \langle E^{s_1}, <^{s_1}, \#^{s_1}, l^{s_1} \rangle$ 和 $S_2 = \langle E^{s_2}, <^{s_2}, \#^{s_2}, l^{s_2} \rangle$ 是一个事件结构中的两个子结构

输出： true 表示两子结构 $S_1, S_2$ 同构

false 表示两子结构 $S_1, S_2$ 不同构

第一步 (判断是否存在双射和是否有相同的动作标记)

**If** $|E^{s_1}| <> |E^{s_2}|$ **then return** false

**If** $L(E^{s_1}) <> L(E^{s_2})$ **then return** false

第二步 (计算原因关系是否保持相同)

**If** $| <^{s_1} | <> | <^{s_2} |$ **then return** false

**While** $| <^{s_1} | \neq 0$

**If** $(e_1, e_1') \in <^{s_1}$ **then**

**if** 存在 $(e_2, e_2') \in <^{s_2}$ 满足 $l(e_1) = l(e_2)$ 和 $l(e_1') = l(e_2')$

**then** $<^{s_1} := <^{s_1} \setminus \{(e_1, e_1')\}$ and $<^{s_2} := <^{s_2} \setminus \{(e_2, e_2')\}$

**else return** false

**End while**

第三步 (计算矛盾关系是否保持相同)

**If** $|\#^{s_1}| <> |\#^{s_2}|$ **then return** false

**While** $|\#^{s_1}| \neq 0$

**If** $e_1 \#^{s_1} e_1'$ **then**

**if** 存在 $(e_2 \#^{s_2} e_2')$ 满足 $l(e_1) = l(e_2)$ 和 $l(e_1') = l(e_2')$

**then** $\#^{s_1} := \#^{s_1} \setminus \{(e_1, e_1'), (e_1', e_1)\}$ 和 $\#^{s_2} := \#^{s_2} \setminus \{(e_2, e_2'), (e_2', e_2)\}$

**else return** false

**End while**

第四步

**Return** true

这个算法是根据事件结构间的同构设计的，所以它是正确的，并且显然能在有限步内完成，算法耗费的时间主要花费在比较两个子结构的原因关系或矛盾关系上，所以时间复杂是 $O(| <^{s_1} | \times | <^{s_2} | + |\#^{s_1}| \times |\#^{s_2}|)$.

上面给出了判断事件结构的两子结构是否同构的算法，现在，可以给出对称约简的算法.

**算法 4.5.2** symmetric-reduction-algorithm( $\mathcal{E}$ )(对称约简算法)

输入： 事件结构 $\mathcal{E} = \langle E, <, \#, l \rangle$

输出： 一个约简的事件结构

第一步

$A := \{e \mid \nexists e' : e' <_{\mathcal{E}} e\}$ ( 获得没有原因关系的事件，即初始出现的事件)

第二步

**If** $|A| = 1$ **then return** $\mathcal{E}$

**else** 将集合 $A$ 划分子集 $A_1, \cdots, A_n (n > 1)$ 并满足以下三个条件

(1) $A_1 \cup \cdots \cup A_n = A$

(2) 对任意 $i, j$ $(i \neq j)$, $A_i \cap A_j = \emptyset$ 和 $\forall e \in A_i, \forall e' \in A_j : e \# e'$

(3) $\forall e_1, e_2 \in A_i : \neg(e_1 \# e_2)$

第三步

计算子结构: $S_1, \cdots, S_n$，其中每个子结构 $S_i$ 对应子集合 $A_i$. 具体计算步骤如下：(对于子集合 $Ai$)

$\#_{\mathcal{E}}(A_i) := \{e' \in E_{\mathcal{E}} \mid \exists e \in A_i : e \#_{\mathcal{E}} e'\}$

$E' := E_{\mathcal{E}} \setminus \#_{\mathcal{E}}(A_i)$

$S_i := \mathcal{E}|_{E'}$

第四步

$S := \{S_1, \cdots, S_n\}$

$S_{\text{quo}} := \{S_1, \cdots, S_n\}$

**While** $|S| > 0$

从集合 $S$ 中选择任意子结构 $S_i$

$S_{\text{temp}} := S \setminus \{S_i\}$

$S := S \setminus \{S_i\}$

**While** $|S_{\text{temp}}| > 0$

从 $S_{\text{temp}}$ 中选择任意子结构 $S_j$

**If call**(Computing-Isomorphism$(S_i, S_j)$)=true

**then** $S_{\text{quo}} := S_{\text{quo}} \setminus \{S_j\}$

$S_{\text{temp}} := S_{\text{temp}} \setminus \{S_j\}$

**End while**

**End while**

第五步 (获取所有剩余事件)

$E'' := \bigcup_{S_i \in S_{\text{quo}}} E^{S_i}$

**Return** $\mathcal{E}|_{E''}$

显然，这个算法是正确的，并且在有限步内终止，它具有多项式时间复杂度，根据前面对算法 4.5.1 的时间复杂度的分析，该算法的时间复杂是 $O(S \times S_{\text{temp}} \times |<^{s_1}| \times |<^{s_2}| + S \times S_{\text{temp}} \times |\#^{s_1}| \times |\#^{s_2}|)$.

注意，这个算法不同于定义 4.2.3 求商事件结构的方法，它只保留同构的子结

构中的一个，使被约简后的事件结构不存在同构的子结构，实现了对称约简的目的. 而定义 4.2.3 中求商事件结构的方法使通过把同构的子结构中互相映射的事件合并在一起，形成事件的轨道，从而求得商事件结构的事件，达到约简的目的. 显然，它们本质上是完全一致的，因为从商事件结构中的每个事件 (原事件的轨道) 中，按原因和独立关系挑出一个代表事件，得到的事件结构与上述算法求出的事件结构是同构的. 下面的例子将说明这一点.

在例 4.2.1 中，给出了求商事件结构的例子，为了比较，这里仍用该例子中的原事件结构，用上述算法计算一次，看看结果如何.

**例 4.5.1** 用例 4.2.1 中的事件结构 $\mathcal{E}_P$ 作为原事件结构. 遵循算法 4.5.1 的步骤，首先得到初始事件作成的集合 $\{e_1, e_2, e_3, e_4, e_5\}$；其次，根据算法 4.5.2, 将事件结构 $\mathcal{E}_P$ 分解成如下三个子结构：

$$S_1 = \langle \{e_1, e_3\}, \emptyset, \emptyset, (l(e_1) = a, l(e_3) = b) \rangle,$$
$$S_2 = \langle \{e_2, e_4\}, \emptyset, \emptyset, (l(e_2) = a, l(e_4) = b) \rangle,$$
$$S_3 = \langle \{e_5, e_6\}, \emptyset, \{(e_5, e_6)\}, (l(e_5) = c, l(e_6) = d) \rangle.$$

最后，因为子结构 $S_1$ 和子结构 $S_2$ 是同构的，这里假定删除子结构 $S_2$，则剩下子结构集合为 $\{S_1, S_3\}$. 最后，获得所有剩下的事件组成的集合 $\{e_1, e_3, e_5, e_6\}$，则约简后的事件结构是 $\langle E_r, <_r, \#_r, l_r \rangle$，其中，

$$E_r = \{e_1, e_3, e_5, e_6\},$$
$$<_r = \{(e_4, e_5)\},$$
$$\#_r = \emptyset,$$
$$l(e_1) = a, l(e_3) = b, l(e_5) = c, l(e_6) = d.$$

该事件结构对应进程 $(a||b) + (c; d)$，比原进程 $P = (a||b) + (a||b) + (c; d)$ 简单.

显然，上述例子中得到的约简后的事件结构与例 4.2.1 中得到的商事件结构是同构的，同构的事件结构在本书中被看成一样的事件结构，所以该算法是正确的.

## 4.6 语法和语义层次上对称约简的重合性

进程代数是一种形式化描述复杂系统的建模工具和高层描述语言，事件结构通常用于定义进程代数的指称语义，它们存在非常密切的关系. 上一章研究了进程代数中进程的对称约简，这里研究事件结构的对称约简，两者分别属于语法和语义层次上的对称约简，它们是否重合，即如果原事件结构是原进程的模型，那么约简后的事件结构是否仍是约简后的进程的模型？这就是本节的讨论内容.

为了研究语法和语义层次上对称约简的重合性，先给出一些定义.

给定一个进程 $\mathcal{P}$，设 $\mathcal{P}_G$ 是进程 $\mathcal{P}$ 的商进程结构，如果 $\mathcal{P}_G$ 不存在对称性，则称 $\mathcal{P}_G$ 是进程 $\mathcal{P}$ 的**最简的商进程结构**.

给定一个事件结构 $\mathcal{E}$，设 $\mathcal{E}_G$ 是事件结构 $\mathcal{E}$ 的商事件结构，如果 $\mathcal{E}_G$ 不存在对称性，则称 $\mathcal{E}_G$ 是进程 $\mathcal{E}$ 的**最简的商事件结构**.

下面给出一个定理.

**定理 4.6.1** 给定一个进程 $\mathcal{P}$，设该进程的事件结构模型是 $\mathcal{E}[\![\mathcal{P}]\!]$，并且设 $\mathcal{P}_G$ 是进程 $\mathcal{P}$ 的最简的商进程结构，$\mathcal{E}[\![\mathcal{P}]\!]_G$ 是 $\mathcal{E}[\![\mathcal{P}]\!]$ 的最简的商事件结构，同时设 $\mathcal{P}_G$ 的事件结构模型是 $\mathcal{E}[\![\mathcal{P}_G]\!]$. 如果进程 $\mathcal{P}$ 表示成结构形式 (定义 3.2.1 给出的形式)，则 $\mathcal{E}[\![\mathcal{P}]\!]_G \simeq_{\mathrm{e}} \mathcal{E}[\![\mathcal{P}_G]\!]$.

**证明** 由于进程的结构定义 3.2.1 与事件结构的定义 2.2.1 是一致的，即 ;, † 和 || 分别对应 $<$、# 和 co，所以当事件结构为结构形式的进程定义语义时，如果在事件结构中显式考虑事件的独立关系 co，则这里的同态映射就是同构映射；同时因为进程的对称性定义 3.2.4 与事件结构的对称性定义 4.2.2 也是一致的，即 ;, † 和 || 分别对应 $<$、# 和 co，所以二者的对称约简也完全一致，如果它们完全去掉对称性，即得到最简的商进程结构 $\mathcal{P}_G$ 和最简的商事件结构是 $\mathcal{E}[\![\mathcal{P}]\!]_G$，则 $\mathcal{P}_G$ 和 $\mathcal{E}[\![\mathcal{P}]\!]_G$(这里显式考虑事件的独立关系) 同构. 又因为事件结构为结构形式的进程定义语义是同构映射，所以 $\mathcal{E}[\![\mathcal{P}]\!]_G \simeq_{\mathrm{e}} \mathcal{E}[\![\mathcal{P}_G]\!]$.

上述定理表明语法和语义层次上对称约简重合. 需要说明的是，上述情况没有考虑进程的递归算子，并且需要进程表示成结构形式才能得出该结论，显然存在局限性. 本书研究进程的对称性和事件结构的对称性都是建立在置换群的基础上，这可能是造成这种局限性的根本原因，我们相信如果直接利用自同构，不考虑利用置换群，并把进程的递归算子纳入其中统一考虑，则问题能够圆满解决.

## 4.7 小　结

本章定义了事件结构对称性的概念，给出了如何得到对称约简后的商事件结构的方法，得出了商事件结构与原事件结构交织迹、交织互模拟和偏序多集迹等价的结论. 这方面的结果说明对称约简充分保留了原事件结构的行为和结构. 与此同时，也证明了商事件结构与原事件结构的交织迹、交织互模拟和偏序多集迹等价在动作细化下是保持的，这再次表明对称约简可以很好地保留原事件结构的性质，也就是说，即使在模型检验中，对称约简也可能是较好的约简算法. 在此工作的基础上，建立了对称约简算法，并举例说明了算法的正确性.

对于事件结构，Penczek[77] 给出了基于偏序约简 (partial order reduction) 实现模型检验的方法，考虑如何将对称约简运用到这种模型检验方法中以进一步缓解状态爆炸问题是值得进一步研究的问题.

# 第五章　对称与自互模拟

## 5.1　引　　言

众所周知，最著名的行为等价是 Milner [70] 和 Park [73] 的互模拟等价 (bisimulation equivalence). 直观地说，两个系统互模拟等价是指无论任何时候它们执行相同的动作序列并到达互模拟等价状态. 互模拟等价已经被证明是与一些结构模型一起用于描述并发和非确定性系统的基础工具. 这里所说的互模拟等价实际上是前面定义 3.4.4 和定义 4.3.4 中定义的“交织互模拟等价”. 由于模型的不同，互模拟等价的定义也有所差别，前面的定义 4.3.4 给出了针对事件结构模型的互模拟等价的概念，认为两个事件结构互模拟等价当且仅当这两个事件结构之间存在一个交织互模拟关系. 这个定义仍用于本章. 也就是说，除非特别声明，本章所提到的互模拟等价实际上是“交织”互模拟等价.

一般地，在相同的事件结构之间存在一种互模拟关系，则这种互模拟称为“自互模拟”(autobisimulation). 很容易证明最大的自互模拟是该事件结构的事件集合上的一种等价关系. 显然，由这种自互模拟关系诱导出的 **互模拟商模型** 互模拟等价于原模型. 本章研究自互模拟约简和对称约简的区别和联系. 给定一个事件结构，由对称约简得到的对称商事件结构与由自互模拟约简得到的互模拟商模型存在许多不同之处. 前者与原事件结构是互模拟等价并偏序多集迹等价的，而且在动作细化下前者能保留原事件结构的行为. 然而，后者不一定是偏序多集迹等价的，同时在动作细化下也不一定能保留原事件结构的行为. 二者相同之处在于，它们都与原事件结构交织迹和 (交织) 互模拟等价，并且如果原事件结构的事件之间不存在独立关系，则它们是同构关系.

## 5.2　自互模拟

前面的定义 4.3.4 给出了针对事件结构模型的互模拟等价的概念，这里给出事件结构“自互模拟”的定义.

**定义 5.2.1**(自互模拟)　设事件结构 $\mathcal{E} \in E$. 如果关系 $R \subseteq C(\mathcal{E}) \times C(\mathcal{E})$ 是在相同的事件结构 $\mathcal{E}$ 中的一种互模拟，那么称 $R$ 是事件结构 $\mathcal{E}$ 的一个**自互模拟**.

因此，给定一个事件结构 $\mathcal{E}$，则最大的自互模拟 $R_m$ 是在 $\mathcal{E}$ 上所有自互模拟的并，即 $R_m = \bigcup\{\, R \mid R$ 是 $\Sigma$ 的一个自互模拟 $\}$. 这里，很容易证明 $R_m$ 是一种等价关系，这种等价关系能诱导出事件结构 $\mathcal{E}$ 的一种商模型. 事实上，这种商模型对应的不是事件结构 $\mathcal{E}$ 本身，而是展示该事件结构行为的标记变迁系统 $\mathrm{LTS}_{\mathcal{E}}$(请参考

4.3 节的叙述). 这里称标记变迁系统 $\mathrm{LTS}_{\mathcal{E}}$ 是事件结构 $\mathcal{E}$ 的**行为模型**. 有了这些解释，可以定义自互模拟约简获得的事件结构的**互模拟商模型**.

**定义 5.2.2**(互模拟商模型) 设事件结构 $\mathcal{E} \in E$，其中 $\mathrm{Act}_{\mathcal{E}}(\mathrm{Act}_{\mathcal{E}} \subseteq \mathrm{Act})$ 是 $\Sigma$ 涉及的动作集合，设该事件结构是由标记变迁系统 $\mathrm{LTS}_{\mathcal{E}} = \langle C(\mathcal{E}), \mathrm{Act}_{\mathcal{E}}, T_{\mathcal{E}}, \varnothing \rangle$ 展示，并且设 $R_m$ 是在标记变迁系统 $\mathrm{LTS}_{\mathcal{E}}$ 上的最大自互模拟，则事件结构 $\mathcal{E}$ 的**行为商模型** 是 $\mathrm{LTS}_{\mathcal{E}_{\mathcal{A}}} = \langle S_A, L_A, T_A, s_{0_A} \rangle$，其中，

- $S_A$ 是 $R_m$ 等价类的集合：$S_A = \{[X] \mid X \in C(\mathcal{E})\}$，其中，在关系 $R_m$ 元素 $X \in C(\mathcal{E})$ 的 $R_m$ 等价类表示为 $[X]$，即 $[X] = \{Y \mid (X,Y) \in R_m\}$；
- $L_A = \mathrm{Act}_{\mathcal{E}}$；
- $T_A = \{([X], a, [X']) \mid X \xrightarrow{a}_{\mathcal{E}} X'\}$；
- $s_{0_A} = \{\varnothing\}$.

一个事件结构称作事件结构 $\mathcal{E}$ 的**互模拟商模型** (表示为 $\mathcal{E}_A$) 当且仅当它的行为由行为商模型 $\mathrm{LTS}_{\mathcal{E}_{\mathcal{A}}}$ 展示.

现在知道，互模拟商模型是一个事件结构的行为模型的约简模型，而定义 4.2.3 给出的商事件结构是事件结构的结构模型. 因此，二者的比较很有意义. 因为行为模型可能损害了事件结构中事件之间的独立关系，所以它们肯定有所不同.

**例 5.2.1** 进程 $Q = (a||b) + (a;b)$ 对应事件结构 $\mathcal{E}_Q$，该事件结构有四个事件：$e_1', e_2', e_3', e_4'$，标记关系为 $l(e_1') = l(e_3') = a, l(e_2') = l(e_4') = b$，原因关系 $e_3' < e_4'$ 并且事件 $e_1', e_2'$ 中的每个都与事件 $e_3', e_4'$ 矛盾. 事件结构 $\mathcal{E}_Q$ 图形化表示如下：

$$\begin{array}{ll} & e_1' \\ & \# \\ \mathcal{E}_Q: & e_3' \longrightarrow e_4' \\ & \# \\ & e_2' \end{array}$$

事件结构 $\mathcal{E}_Q$ 可能的格局是 $\varnothing, \{e_1'\}, \{e_2'\}, \{e_3'\}, \{e_1', e_2'\}$ 和 $\{e_3', e_4'\}$. 它的行为由标记变迁系统 $\mathrm{LTS}_{\mathcal{E}_Q} = \langle C(\mathcal{E}_Q), \mathrm{Act}_{\mathcal{E}_Q}, T_{\mathcal{E}_Q}, \varnothing \rangle$ 展示：

- $C(\mathcal{E}_Q) = \{\varnothing, \{e_1'\}, \{e_2'\}, \{e_3'\}, \{e_1', e_2'\}, \{e_3', e_4'\}\}$；
- $\mathrm{Act}_{\mathcal{E}_Q} = \{a, b\}$；
- $T_{\mathcal{E}_Q} = \{\ (\varnothing, a, \{e_1'\}), (\varnothing, b, \{e_2'\}), (\{e_1'\}, b, \{e_1', e_2'\}), (\{e_2'\}, a, \{e_1', e_2'\}), (\varnothing, a, \{e_3'\}), (\{e_3'\}, b, \{e_3', e_4'\})\ \}$.

标记变迁系统 $\mathrm{LTS}_{\mathcal{E}_Q}$ 的变迁关系描述在图 5.1 中.

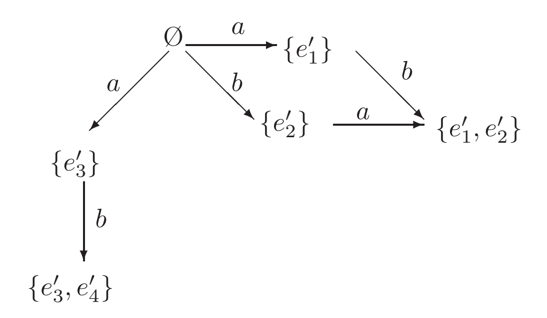



图 5.1 $\mathrm{LTS}_{\mathcal{E}_Q}$ 的变迁关系

显然，事件结构 $\mathcal{E}_Q$ 的最大自互模拟 $R_m = \{(\emptyset, \emptyset),(\{e'_1\},\{e'_1\}), (\{e'_2\},\{e'_2\}), (\{e'_3\},\{e'_3\}), (\{e'_1\},\{e'_3\}), (\{e'_3\},\{e'_1\}), (\{e'_1, e'_2\},\{e'_3, e'_4\}), (\{e'_3, e'_4\},\{e'_1, e'_2\}), (\{e'_1, e'_2\},\{e'_1, e'_2\}),(\{e'_3, e'_4\},\{e'_3, e'_4\})\ \}$. 根据上述定义，图 5.2 描述了事件结构 $\mathcal{E}_Q$ 的行为商模型 $\langle S_A, L_A, T_A, s_{0_A}\rangle$：

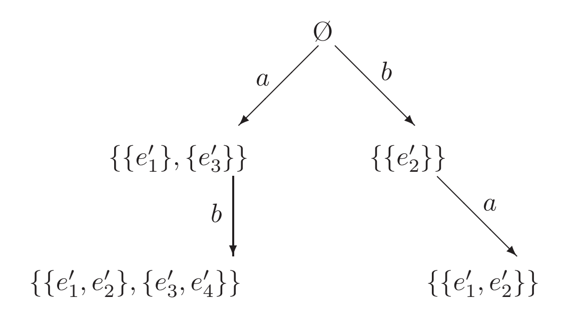



图 5.2 $\mathrm{LTS}_{\mathcal{E}_{QA}}$ 的变迁关系

- $S_A = \{\ \{\emptyset\}, \{\{e'_2\}\}, \{\{e'_1\},\{e'_3\}\}, \{\{e'_1, e'_2\},\{e'_3, e'_4\}\}\ \}$;
- $L_A = \{a, b\}$;
- $T_A = \{\ (\emptyset, a, \{\{e'_1\},\{e'_3\}\}), (\{\{e'_1\},\{e'_3\}\}, b, \{\{e'_1, e'_2\},\{e'_3, e'_4\}\}), (\emptyset, b, \{\{e'_2\}\}), (\{\{e'_2\}\}, a, \{\{e'_1, e'_2\}\})\ \}$;
- $s_{0_A} = \{\emptyset\}$.

因为上述互模拟约简不考虑事件结构 $\mathcal{E}_Q$ 中事件之间的独立关系，所以事件结构 $\mathcal{E}_Q$ 的互模拟商模型是事件结构 $\mathcal{E}_{A1}$ 或是事件结构 $\mathcal{E}_{A2}$，它们被描述在图 5.3 中. 显然，事件结构 $\mathcal{E}_{A1}$ 对应进程 $a; b + b; a$，而事件结构 $\mathcal{E}_{A2}$ 对应进程 $a||b$.

$$\mathcal{E}_{A1}: \quad \begin{array}{c} e_3'' \longrightarrow e_4'' \\ \# \\ e_2'' \longrightarrow e_1'' \end{array} \qquad\qquad \mathcal{E}_{A2}: \quad \begin{array}{c} e_1'' \\ \\ e_2'' \end{array}$$

图 5.3 两个事件结构：$\mathcal{E}_{A1}$ 和 $\mathcal{E}_{A2}$

根据上述定义，直接有下面的结论.

**定理 5.2.1** 设事件结构 $\mathcal{E} \in E$，设事件结构 $\mathcal{E}$ 的行为由标记变迁系统 $\mathrm{LTS}_{\mathcal{E}}$ 展示，并且设标记变迁系统 $\mathrm{LTS}_{\mathcal{E}_A}$ 是事件结构 $\mathcal{E}$ 的行为商模型，而且，设 $\mathcal{E}_A$ 是事件结构 $\mathcal{E}$ 的互模拟商模型. 则有 $\mathcal{E} \approx_{\mathrm{ib}} \mathcal{E}_A$.

证明根据定义直接可得.

交织互模拟等价能直接推出交织迹等价，所以下面结论成立.

**定理 5.2.2** 设事件结构 $\mathcal{E} \in E$，设事件结构 $\mathcal{E}$ 的行为由标记变迁系统 $\mathrm{LTS}_{\mathcal{E}}$ 展示，并且设标记变迁系统 $\mathrm{LTS}_{\mathcal{E}_A}$ 是事件结构 $\mathcal{E}$ 的行为商模型，而且，设 $\mathcal{E}_A$ 是事件结构 $\mathcal{E}$ 的互模拟商模型. 则有 $\mathcal{E} \approx_{\mathrm{it}} \mathcal{E}_A$.

证明根据定义直接可得.

## 5.3 自互模拟与对称的区别

这部分从两个方面来揭示自互模拟与对称的区别：(1) 等价的保持；(2) 动作细化下行为的保持. 先给出一个例子.

**例 5.3.1** 这里继续考虑例 4.2.1. 很明显，在例 4.2.1 中的事件结构 $\mathcal{E}_P$ (交织) 互模拟等价于例 5.2.1 中的事件结构 $\mathcal{E}_Q$ ：$\mathcal{E}_Q \approx_{\mathrm{ib}} \mathcal{E}_Q$. 因此，事件结构 $\mathcal{E}_P$ 的互模拟商模型就是例 5.2.1 中的事件结构 $\mathcal{E}_Q$ 的互模拟商模型：事件结构 $\mathcal{E}_{A1}$ 或事件结构 $\mathcal{E}_{A2}$，它们被描述在图 5.3 中. 而例 4.2.3 中的事件结构 $\mathcal{E}_P$ 的商事件结构 $\mathcal{E}_{PG} = \langle E_G, <_G, \#_G, l_G \rangle$:

$E_G = \{\ \{e_1, e_2\}, \{e_3, e_4\}, \{e_5\}, \{e_6\}\ \}$,

$<_G = \{\ (\{e_5\}, \{e_6\})\ \}$,

$\#_G = \{\ (\{e_1, e_2\}, \{e_5\}), (\{e_5\}, \{e_1, e_2\}), (\{e_5\}, \{e_3, e_4\}), (\{e_3, e_4\}, \{e_5\}), (\{e_1, e_2\}, \{e_6\}), (\{e_6\}, \{e_1, e_2\}), (\{e_3, e_4\}, \{e_6\}), (\{e_6\}, \{e_3, e_4\})\ \}$,

$l_G(\{e_1, e_2\}) = a, l_G(\{e_3, e_4\}) = b, l_G(\{e_5\}) = c, l_G(\{e_6\}) = d.$

商事件结构 $\mathcal{E}_{PG}$ 被描述在下面：

$$\mathcal{E}_{PG}: \quad \begin{array}{c} \{e_1, e_2\} \\ \# \\ \{e_5\} \longrightarrow \{e_6\} \\ \# \\ \{e_3, e_4\} \end{array}$$

$\mathcal{E}_{PG}$ 恰恰与事件结构 $\mathcal{E}_Q$ 同构: $\mathcal{E}_{PG} \simeq_{\mathrm{e}} \mathcal{E}_Q$ .

在上例中, 如果选择事件结构 $\mathcal{E}_{A1}$ 作为事件结构 $\mathcal{E}_P$ 的互模拟商模型, 则有 (1) $\mathcal{E}_P \approx_{\mathrm{it}} \mathcal{E}_{A1}$; (2)$\mathcal{E}_P \approx_{\mathrm{ib}} \mathcal{E}_{A1}$ 和 (3)$\mathcal{E}_P \approx_{\mathrm{pt}} \mathcal{E}_{A1}$. 也就是说, 原事件结构与互模拟商模型交织迹等价、交织互模拟等价并且偏序多集迹等价. 但是, 如果选择事件结构 $\mathcal{E}_{A2}$ 作为事件结构 $\mathcal{E}_P$ 的互模拟商模型, 则有 (1) $\mathcal{E}_P \approx_{\mathrm{it}} \mathcal{E}_{A2}$, (2)$\mathcal{E}_P \approx_{\mathrm{ib}} \mathcal{E}_{A2}$, (3)$\mathcal{E}_P \not\approx_{\mathrm{pt}} \mathcal{E}_{A2}$, 即原事件结构与互模拟商模型交织迹等价、交织互模拟等价, 但是不偏序多集迹等价. 因此自互模拟约简与对称约简在等价上存在这样的差别：偏序多集迹等价在自互模拟约简下可能不保持.

另一方面，我们研究在动作细化下对称与互模拟等价的差别. 在上例中，如果选择事件结构 $\mathcal{E}_{A2}$ 作为事件结构 $\mathcal{E}_P$ 的互模拟商模型，$\mathcal{E}_P \not\approx_{\mathrm{pt}} \mathcal{E}_{A2}$，设 ref 是一个细化函数，则有 $\mathrm{ref}(\mathcal{E}_{A1}) \not\approx_{\mathrm{pt}} \mathrm{ref}(\mathcal{E}_Q)$ . 另外，文献 [41] 中的例 6.1 ：两系统 $P_1 = a||b$，$P_2 = a;b+b;a$. 如果在相同的细化函数 ref 作用下，将动作 $a$ 细化成 $a_1;a_2$，而动作 $b$ 保持不变，则在交织语义下，细化后的进程为 $P_1' = a_1;a_2||b = a_1;\ a_2;\ b+a_1;\ b;\ a_2+b;\ a_1;\ a_2$ (参见图 5.4(a))，$Q_1' = a_1;\ a_2;\ b+b;\ a_1;\ a_2$ (参见图 5.4(b)).

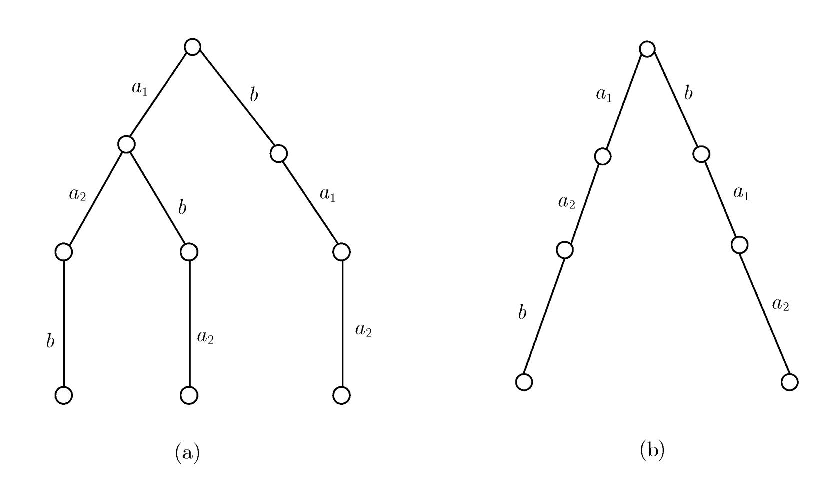



图 5.4 在交织语义下的动作细化

根据文献 [41] 中的例 6.1 ，显然有 $\mathrm{ref}(\mathcal{E}_{A1}) \not\approx_{\mathrm{it}} \mathrm{ref}(\mathcal{E}_Q)$ ，并且有 $\mathrm{ref}(\mathcal{E}_{A1}) \not\approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{E}_Q)$. 因此，事件结构与其互模拟商模型在动作细化下等价不保持. 而定理 2.3.1 表明事件结构与其商事件结构在动作细化下等价是保持的. 这就是对称和自互模拟的第二点区别.

至于对称约简与自互模拟约简的其他区别，本书不再过多论述，需要说明的是，毕竟对称约简是结构约简，而自互模拟约简是行为约简，从根本上来讲这种差别是必然的.

## 5.4 自互模拟与对称的联系

上节讨论了自互模拟与对称的区别，本节将讨论它们之间的联系. 首先需要说明的是，根据定理 4.3.1 和定理 4.3.2，以及本章的定理 5.2.1 和定理 5.2.2 可知：给定一个事件结构，由对称约简得到的对称商事件结构与由自互模拟约简得到的互模拟商模型与原事件结构都是交织迹和交织互模拟等价的. 这是自互模拟与对称的联系之一.

接下来讨论自互模拟与对称的另一方面联系. 从上一节可知，可能是由于自互模拟约简破坏了事件结构中的独立关系，才导致了与对称约简的不同. 这就使人们很自然地想到了一个问题: 如果事件结构中的事件之间不存在独立关系，那么情况会怎样呢？

为了回答这个问题，首先给出下面的定理.

**定理 5.4.1** 设事件结构 $\mathcal{E} \in E$, 并设 $\mathcal{E}_A$, $\mathcal{E}_G$ 分别是事件结构 $\mathcal{E}$ 的互模拟商模型和最简的商事件结构. 如果事件结构的独立关系为空，即 $\mathrm{co}_{\mathcal{E}} = \varnothing$, 则 $\mathcal{E}_A \simeq_{\mathrm{e}} \mathcal{E}_G$.

**证明** 设 $R_m$ 是事件结构 $\mathcal{E}$ 的最大自互模拟，并设任意两个不同的格局 $X, X'$, 并满足 $[X] = [X']$(属于相同的等价类)，由于 $\mathrm{co}_{\mathcal{E}} = \varnothing$, 所以在事件结构 $\mathcal{E}$ 中的每个格局中的事件之间的关系是一个全序原因关系，因此可以设事件 $e, e'$ 分别是格局 $X, X'$ 的最大事件. 由于格局 $X$ 和 $X'$ 是互模拟到达的互模拟状态，根据其全序原因关系，能构造出一个双射：$f$ ，一部分是 $X \to X'$ 使得它们按全序关系对应，其中对应的事件显然保持相同的原因、矛盾关系和相同的标记; 另一部分是一部分未用到的事件，即剩余的事件，让它们自己对应自己，这样构成的双射一定是一个自同构. 因此，每一个格局对应一个该格局中的最大事件，相应地，一个互模拟的格局的等价类，对应于该格局最大事件的轨道，所以由于没有事件之间的独立关系，自互模拟约简等同于对称约简. 这个结论显然成立.

上述定理表明，对称约简与自互模拟约简在一定条件下是一致的.

本节讨论了自互模拟与对称两者间的联系，可以肯定的是还存在其他方面的联系，此处不再赘述. 我们工作的目的是希望从结构上建立类似于行为等价的由粗

到细的约简体制，为设计开发并发系统而建立不同层次的静态的约简模型服务，这些研究已经足够了.

## 5.5 小 结

自互模拟约简是行为约简，而对称约简是结构约简. 它们既有区别，又有联系. 区别在于，自互模拟约简可能不保持偏序多集迹等价，并且在动作细化下交织迹、交织互模拟和偏序多集迹等价都不一定保持；而对称约简保持了交织迹、交织互模拟和偏序多集迹等价，以及在动作细化下这三种等价都保持. 这些区别，从另一方面也说明了自互模拟约简比对称约简得到的约简后的模型要小，但保留的性质也较少. 因此，这两种约简可能用在处理不同的问题中. 我们认为，如果用在结构层次的模型中，应该首先进行对称约简，在对称约简的基础上再考虑其他约简. 特别是，在处理状态爆炸等问题时，尤其需要考虑这点. 当然，自互模拟约简和对称约简也存在一致的地方，即在事件结构不存在独立关系的事件时，它们约简后得到的事件结构是同构的. 这也反过来说明自互模拟约简打破了独立事件间的独立关系，才导致了它与对称约简的不同. 这些研究有利于人们看清结构约简和行为约简的差别，从而开发出更好更合适的约简方法[54].

本书中涉及到的由粗到细的行为等价有如下关系：

$$\text{交织迹等价} \supseteq \text{交织互模拟等价} \supseteq \text{保持历史互模拟等价} \supseteq \text{同构}.$$

显然，由这些等价建立的对应的约简方法就构成了一种由粗到细的行为约简体系：

$$\text{交织迹约简} \Longrightarrow \text{交织互模拟约简} \Longrightarrow \text{保持历史互模拟约简} \Longrightarrow \text{对称约简}.$$

很明显，对称约简是最细的行为约简，也是最细的结构 (静态) 约简. 对一个给定的复杂系统，可以根据不同的需要建立不同大小的约简模型，从而为分析、设计和开发系统服务. 但是，这些经约简得到的模型是行为模型，由于动态行为非常复杂，约简问题很可能变得困难. 相对来说，静态约简可能不考虑系统的动态行为，操作起来更容易，所以，一个值得进一步研究的课题是，从结构上建立起类似于行为等价的由粗到细的约简体系.

# 第六章　等价在动作细化下的保持

## 6.1 引　　言

在对并发系统形式化建模的过程中，系统地研究语义等价有利于更好地理解并发系统的重要性质. 对于并发系统的设计开发，希望有支持层次结构的形式化方法，在这种自上而下的设计开发过程中，系统将从抽象的刻画逐步进化成更具体的实现. 动作细化是系统层次化刻画方法的核心操作，这种细化技术以分层的方式来描述系统，也就是说，用较低抽象层次上复杂的进程来解释较高抽象层次上的动作，从而改变其抽象层次，最终达到实现层次. 人们在进程代数及其语义模型中已经对这种方法进行了大量研究. 然而，又存在一个重要的问题：如果两个进程是等价的，人们希望这两个进程在动作被细化后，得到的两个进程仍然等价，即动作细化下等价的保持问题. 比如对某一类等价 $\sim$，给定两个进程 $P$ 和 $Q$，如果 $P \sim Q$，并且 ref 是一个动作细化函数，那么下面的情况

$$\mathrm{ref}(P) \sim \mathrm{ref}(Q)$$

是否成立？在线型和分支时间等价谱系 [36] 中，人们已经知道交织等价 (即交织迹等价 (interleaving trace equivalence) 和交织互模拟等价 (interleaving bisimulation equivalence)) 与步进等价 (即步进迹等价 ( step trace equivalence) 和步进互模拟等价 ( step bisimulation equivalence)) 在动作细化下是不保持的 [24,41]. 一些工作集中在限制动作细化，使得这些等价在限制的条件下是保持的，并且只证明了交织互模拟等价在非常严格的限制条件下才能保持，但在这种限制下交织迹等价仍然不保持 [24]. 另外，没有工作进一步讨论步进迹等价和步进互模拟等价动作细化下的保持问题. 本章定义了一类具有特定性质的并发进程，它们能使交织等价和步进等价在没有限制动作细化的条件下是保持的. 也就是说，在没有原因的独立关系的并发进程中，或者在特定并发环境 (进程的所有变迁关系都是“束动作”变迁) 的情况下，这两类交织等价和两类步进等价被证明在动作细化下是保持的. 其中，我们提出了“束动作变迁”的概念，这种处理系统变迁的方式不同于基于迹理论采用“偏序约简”以减少并发系统的状态，而是将完全并发的动作看成一个“大动作”，使得系统在这个大动作执行的前后仅存在两个状态，因此，束动作变迁的概念能用于处理系统验证过程中状态爆炸的问题，在下一章将进一步讨论怎样在模型检验中实现这种处理动作变迁的方法.

Vogler [88,89] 最先提出了处理动作细化下等价保持问题的基本思想. Czaja, van Glabbeek 和 Goltz [24] 证明了交织互模拟如果在动作细化后不出现选择运算和

动作自并发情况，则交织互模拟等价在动作细化下是保持的，但是交织迹等价仍然不保持. Goltz 和 Wehrheim [45] 证明了带有保持历史交织互模拟 (history preserving bisimulation) 与具有全局依赖关系的原因测试关系是一致的，但是他们没有更进一步讨论动作细化下的保持问题，并且没有讨论在动作独立关系环境下将会出现的其他情况. 最后，van Glabbeek 和 Goltz 在文献 [41] 中总结了最近十多年来有关动作细化的研究成果，对动作细化下的等价保持问题作出了较详细解释，并证明了交织等价在一般的动作细化下不保持，但也没有更进一步讨论. 文献 [53] 扩展了这些工作，重点讨论了步进等价在动作细化下的保持问题.

## 6.2 交织等价

在本书的 §4.3 中已经引入了基于事件结构建立起来的交织迹等价、交织互模拟等价和偏序多集迹等价概念 [41]，此处，为了论述的完整性，再次给出定义，以方便读者. 另外，为了方便，本节还引入了保持历史等价 (history preserving equivalence) 的概念. 在本节的最后，给出一个结论：如果事件结构不存在原因独立关系，则交织迹和交织互模拟等价在动作细化下是保持的.

为了引入交织等价，先给出 “单个动作变迁” 的定义.

**定义 6.2.1**(单个动作变迁) 设事件结构 $\mathcal{E} \in E$. 一个变迁 $X \xrightarrow{a}_{\mathcal{E}} X'$ 称作**单个动作变迁**，当且仅当 $a \in \mathrm{Act}, X, X' \in C(\mathcal{E}), X \subseteq X'$，并且 $\exists e \in E_{\mathcal{E}}$: $X' - X = e(l_{\mathcal{E}}(e) = a)$.

这里，$X \xrightarrow{a}_{\mathcal{E}} X'$ 表示在事件结构 $\mathcal{E}$ 中，通过执行动作 $a$，格局 $X$ 表示的状态可能进化成 $X'$ 表示的状态. 这个变迁关系关联着一个标记变迁系统 (定义 2.4.1) 和一个事件结构. 因此，定义在标记变迁系统上的等价直接诱导出定义在事件结构上对应的等价.

为了定义交织迹等价，需要先给出 “迹” 的定义.

**定义 6.2.2**(迹) 设事件结构 $\mathcal{E} \in E$. 一个字 $w = a_1 \cdots a_n \in \mathrm{Act}^*$ 是事件结构 $\mathcal{E}$ 的一个**迹**当且仅当 $\exists X_0, \cdots, X_n \in C(\mathcal{E}) : X_0 = \emptyset$ 和 $X_{i-1} \xrightarrow{a_i} X_i, i = 1, \cdots, n$.

这里，$\mathrm{tr}s(\mathcal{E})$ 表示事件结构 $\mathcal{E}$ 的所有迹的集合. 接下来，定义交织迹等价.

**定义 6.2.3**(交织迹等价) 设事件结构 $\mathcal{E}, \mathcal{F} \in E$. $\mathcal{E}$ 和 $\mathcal{F}$ 称作**交织迹等价**(表示为 $\mathcal{E} \approx_{\mathrm{it}} \mathcal{F}$ ) 当且仅当 $\mathrm{tr}s(\mathcal{E}) = \mathrm{tr}s(\mathcal{F})$.

接下来，引入交织互模拟等价的概念 [41]. 注意，人们可能直接称其为 “互模拟等价”，没有 “交织” 这两个字. 但根据文献 [41]，称 “交织互模拟等价” 更为准确.

**定义 6.2.4** 设两事件结构 $\mathcal{E}, \mathcal{F} \in E$. 一个关系 $R \subseteq C(\mathcal{E}) \times C(\mathcal{F})$ 称作 $\mathcal{E}$ 和 $\mathcal{F}$ 之间的一个**交织互模拟**当且仅当 $(\emptyset, \emptyset) \in R$，并且如果 $(X, Y) \in R$，那么

- $X \xrightarrow{a}_{\mathcal{E}} X', a \in \mathrm{Act} \Rightarrow \exists Y' : Y \xrightarrow{a}_{\mathcal{F}} Y' \wedge (X', Y') \in R$,
- $Y \xrightarrow{a}_{\mathcal{F}} Y', a \in \mathrm{Act} \Rightarrow \exists X' : X \xrightarrow{a}_{\mathcal{E}} X' \wedge (X', Y') \in R$.

$\mathcal{E}$ 与 $\mathcal{F}$ 称作**交织互模拟等价** (表示为 $\mathcal{E} \approx_{\mathrm{ib}} \mathcal{F}$ )，当且仅当 $\mathcal{E}$ 和 $\mathcal{F}$ 之间存在一个交织互模拟.

显然，如果不考虑终止敏感的变迁 (终止敏感的变迁是指可能不完全的终止)，则有如下结论：

$$\mathcal{E} \approx_{\mathrm{ib}} \mathcal{F} \Rightarrow \mathcal{E} \approx_{\mathrm{it}} \mathcal{F}.$$

交织迹等价和交织互模拟等价都是基于交织模型的，即基于单个动作变迁的标记变迁系统. 这两类等价是两类重要的交织等价，研究它们有重要的意义. 最后，引入另外一类等价，它是基于偏序执行建立起来的，称作“偏序多集迹等价 (pomset trace equivalence)” [41]. 我们还将研究商事件结构和原事件结构是否偏序多集迹等价.

**定义 6.2.5** 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$, 并设 $\mathrm{pomsets}(\mathcal{E}) = \{\mathrm{pomset}(X) \mid X \in C(\mathcal{E})\}$ 和 $\mathrm{pomsets}(\mathcal{F}) = \{\mathrm{pomset}(Y) \mid Y \in C(\mathcal{F})\}$. $\mathcal{E}$ 和 $\mathcal{F}$ 称作**偏序多集迹等价**(表示为 $\mathcal{E} \approx_{\mathrm{pt}} \mathcal{F}$), 当且仅当 $\mathrm{pomsets}(\mathcal{E}) = \mathrm{Pomsets}(\mathcal{F})$.

文献 [41] 证明了偏序多集迹等价在动作细化下是保持的，为了后面论述的需要，引入下面的定理.

**定理 6.2.1** 如果两个事件结构 $\mathcal{E}, \mathcal{F} \in E$, ref 是一个细化函数，并且 $\mathcal{E} \approx_{\mathrm{pt}} \mathcal{F}$, 则 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{pt}} \mathrm{ref}(\mathcal{F})$.

证明参见文献 [41] 中的定理 8.1.

偏序多集迹等价要求事件之间的原因、独立关系都一致，显然可以直接得出它蕴含交织迹等价的结论.

**命题 6.2.1** 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$, 则有 $\mathcal{E} \approx_{\mathrm{pt}} \mathcal{F} \Rightarrow \mathcal{E} \approx_{\mathrm{it}} \mathcal{F}$.

证明根据上述定义可以直接得出.

接下来，又引入一类等价概念：保持历史等价 (history preserving bisimulation)，它是比交织互模拟更细的一类等价, 它在动作细化下是保持的 [41].

**定义 6.2.6**(保持历史等价) 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$. 关系 $R \subseteq C(\mathcal{E}) \times C(\mathcal{F}) \times \mathcal{P}(E_{\mathcal{E}} \times E_{\mathcal{F}})$ 称作两个事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 之间的一个**保持历史互模拟**当且仅当 $(\emptyset, \emptyset, \emptyset) \in R$, 并且无论任何时候 $(X, Y, f) \in R$ ，则有

- $f$ 是 $X$ 和 $Y$ 之间的一个双射，
- $X \xrightarrow{a}_{\mathcal{E}} X', a \in \mathrm{Act} \Rightarrow \exists Y', f' : Y \xrightarrow{a}_{\mathcal{F}} Y', (X', Y', f') \in R \wedge f'|_X = f$,
- $Y \xrightarrow{a}_{\mathcal{F}} Y', a \in \mathrm{Act} \Rightarrow \exists X', f' : X \xrightarrow{a}_{\mathcal{E}} X', (X', Y', f') \in R \wedge f'|_X = f$.

事件结构 $\mathcal{E}$ 和 事件结构 $\mathcal{F}$ 称作**保持历史等价**(表示为 $\mathcal{E} \approx_{\mathrm{h}} \mathcal{F}$)，当且仅当 $\mathcal{E}$ 和 $\mathcal{F}$ 之间存在一个保持历史互模拟.

保持历史等价在动作细化下是保持的，下面以定理的形式给出这一结论.

**定理 6.2.2** 如果两个事件结构 $\mathcal{E}, \mathcal{F} \in E$，并且 ref 是一个细化函数，则 $\mathcal{E} \approx_{\mathrm{h}} \mathcal{F} \Rightarrow \mathrm{ref}(\mathcal{E}) \approx_{\mathrm{h}} \mathrm{ref}(\mathcal{F})$.

证明参见文献 [41] 中的定理 9.1 和命题 9.2.

根据定义 6.2.4 和定义 6.2.6，直接可以得出保持历史等价蕴含交织互模拟等价.

**命题 6.2.2** 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$，则 $\mathcal{E} \approx_{\mathrm{h}} \mathcal{F} \Rightarrow \mathcal{E} \approx_{\mathrm{ib}} \mathcal{F}$.

直接按定义加以证明即可.

交织等价在动作细化下不保持，然而，如果限制它们基于的事件结构模型的性质，就能得到一些有用的结果. 下面的定理表明，如果事件结构中不存在事件间的独立关系，那么建立在这类事件结构上的交织等价 (交织迹和交织互模拟等价) 在动作细化下是保持的.

**定理 6.2.3** 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$. 如果 $\mathrm{co}_{\mathcal{E}} = \mathrm{co}_{\mathcal{F}} = \emptyset$，并且 ref 是一个细化函数，则

(1) $\mathcal{E} \approx_{\mathrm{it}} \mathcal{F} \Rightarrow \mathrm{ref}(\mathcal{E}) \approx_{\mathrm{it}} \mathrm{ref}(\mathcal{F})$，

(2) $\mathcal{E} \approx_{\mathrm{ib}} \mathcal{F} \Rightarrow \mathrm{ref}(\mathcal{E}) \approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{F})$.

**证明** (1) 对于任何格局 $X \in C(\mathcal{E})$，根据定义 2.2.2，则有 $\#_{\mathcal{E}|_X} = \emptyset$. 而且，由于 $\mathrm{co}_{\mathcal{E}} = \emptyset$ ，并且 $X$ 是左封闭的，因此在格局 $X$ 存在一个全序关系. 在任意格局 $X$ 中的每个事件都被一个动作标记，根据定义 6.2.2，格局 $X$ 对应着唯一条迹 (表示为 $w_X$). 又由于同构存在对应的相同的标记，则有 $\forall X, Y \in C(\mathcal{E}) \wedge X \not\simeq_{\mathrm{p}} Y \Rightarrow \exists w_X, w_Y \in \mathrm{tr}s(\mathcal{E}) \wedge w_X \neq w_Y$. 显然，如果动作名不引起混淆的话，用动作名表示事件名，则 $\mathrm{pomsets}(\mathcal{E}) = \mathrm{tr}s(\mathcal{E})$. 用同样的方法，也有 $\mathrm{pomsets}(\mathcal{F}) = \mathrm{tr}s(\mathcal{F})$. 因为 $\mathcal{E} \approx_{\mathrm{it}} \mathcal{F}$，所以有 $\mathrm{tr}s(\mathcal{E}) = \mathrm{tr}s(\mathcal{F})$. 因此，可以得出 $\mathrm{tr}s(\mathcal{E}) = \mathrm{tr}s(\mathcal{F}) = \mathrm{pomsets}(\mathcal{E}) = \mathrm{pomsets}(\mathcal{F})$，即 $\mathcal{E} \approx_{\mathrm{it}} \mathcal{F} \Rightarrow \mathcal{E} \approx_{\mathrm{pt}} \mathcal{F}$. 同时, 因为定理 6.2.1，则有 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{pt}} \mathrm{ref}(\mathcal{F})$. 而且又因为命题 6.2.1，所以可以得出 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{it}} \mathrm{ref}(\mathcal{F})$.

(2) 根据上面的证明得出, 对于任何格局 $X \in C(\mathcal{E})$ ，仅存在事件间的全序关系，因此对于任意 $Y \in C(\mathcal{F})$ 也是这样. 由于 $\mathcal{E} \approx_{\mathrm{ib}} \mathcal{F}$, 则有 $X \xrightarrow{a}_{\mathcal{E}} X', a \in \mathrm{Act} \Rightarrow \exists Y' : Y \xrightarrow{a}_{\mathcal{F}} Y' \wedge (X', Y') \in R$ 和 $(\emptyset, \emptyset), (X, Y), (X', Y') \in R$ 成立. 现在要证明 $X \simeq_{\mathrm{p}} Y$ ，并且 $X' \simeq_{\mathrm{p}} Y'$. 显然，两事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 之间的一个交织互模拟开始于初始格局 $(\emptyset)$ 之间的关系，并通过执行相同的动作到达互模拟的状态 (设为 $X$ 和 $Y$ ). 根据上面已知 $X$ 和 $Y$ 是全序，并且它们是执行相同动作对应的事件，所以有 $X \simeq_{\mathrm{p}} Y$，同样有 $X' \simeq_{\mathrm{p}} Y'$. 根据对称关系，对于任意 $Y \xrightarrow{a}_{\mathcal{F}} Y', a \in \mathrm{Act} \Rightarrow \exists X' : X \xrightarrow{a}_{\mathcal{E}} X' \wedge (X', Y') \in R$，也有 $Y \simeq_{\mathrm{p}} X$ 和 $Y' \simeq_{\mathrm{p}} X'$. 遵循上述方法, 能找到事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 之间的一个保持历史互模拟, 则有 $\mathcal{E} \approx_{\mathrm{h}} \mathcal{F}$，根据定理 6.2.2，又有 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{h}} \mathrm{ref}(\mathcal{F})$. 因此，根据命题 6.2.2，可以得

出结论 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{F})$.

## 6.3 步 进 等 价

本节我们泛化了来自事件结构的单个动作变迁 $X \xrightarrow{a} X'$, 形成了另一种变迁 $X \xrightarrow{A} X'$(其中 $A$ 是动作集合 Act 上的多重集), 称作“步进动作变迁”[79]. 采用这种变迁, 可以直接根据上一节定义的交织等价泛化为将引入的等价 —— 步进等价. 因此, 首先需要对步进动作变迁作出严格的数学定义, 才能引入步进迹等价 (step trace equivalence) 和步进互模拟等价 (step bisimulation equivalence) 两类步进等价[41]. 另外, 也将讨论这两类等价在动作细化下的保持问题, 特别在本节的最后给出一个有意义的定理.

**定义 6.3.1**(步进动作变迁) 设事件结构 $\mathcal{E} \in E$. 一个变迁 $X \xrightarrow{A} X'$ 称作一个**步进动作变迁**, 当且仅当 $A \in \mathbb{N}^{\mathrm{Act}}$(即 $A$ 是动作集合 Act 上的多重集), $X, X' \in C(\mathcal{E}), X \subseteq X'$ , 并且 $X' - X = G$ , 使得 $\forall d, e \in G : d \ \mathrm{co}_{X'} \ e$ 和 $l_{\mathcal{E}}(G) = A$. 这里, $l_{\mathcal{E}}(G) \in \mathbb{N}^{\mathrm{Act}}$ 是由 $l_{\mathcal{E}}(G)(a) = |\{e \in G | l_{\mathcal{E}}(e) = a\}|$ 给出的.

这里, $X \xrightarrow{A}_{\mathcal{E}} X'$ 表示在事件结构 $\mathcal{E}$ 中, 通过并发执行集合 $A$ 中的所有动作, 格局 $X$ 表示的状态可能进化成 $X'$ 表示的状态. 这个变迁关系关联着一个标记变迁系统 (定义 2.4.1) 和一个事件结构. 因此, 定义在标记变迁系统上的等价直接诱导出定义在事件结构上对应的等价.

根据上述定义, 显然有如下命题.

**命题 6.3.1** 一个单个动作变迁是一个步进动作变迁.

证明从略.

为了引入步进迹等价的概念, 还必须定义**步进迹**(step trace) 这一概念.

**定义 6.3.2**(步进迹) 设事件结构 $\mathcal{E} \in E$. 一个序列 $W = A_1 \cdots A_n (A_i \in \mathbb{N}^{\mathrm{Act}} (i = 1, \cdots, n))$ 称作事件结构 $\mathcal{E}$ 的一个**步进迹**, 当且仅当 $\exists X_0, \cdots, X_n \in C(\mathcal{E}) : X_0 = \emptyset$ 并且 $X_{i-1} \xrightarrow{A_i} X_i (i = 1, \cdots, n)$ 是一个步进变迁.

这里, $\mathrm{steptr}s(\mathcal{E})$ 用于表示事件结构 $\mathcal{E}$ 所有步进迹组成的集合. 有了这些定义和记号, 就可以定义步进迹等价了.

**定义 6.3.3**(步进迹等价) 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$, 以及它们的步进集合 $\mathrm{steptr}s(\mathcal{E})$ 和 $\mathrm{steptr}s(\mathcal{F})$. 事件结构 $\mathcal{E}$ 和事件结构 $\mathcal{F}$ 称作**步进迹等价**(表示为 $\mathcal{E} \approx_{\mathrm{st}} \mathcal{F}$) 当且仅当 $\mathrm{steptr}s(\mathcal{E}) = \mathrm{steptr}s(\mathcal{F})$.

接下来, 定义另一类等价.

**定义 6.3.4**(步进互模拟等价) 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$. 一个关系 $R \subseteq C(\mathcal{E}) \times C(\mathcal{F})$ 称作事件结构 $\mathcal{E}$ 和事件结构 $\mathcal{F}$ 之间的一个**步进互模拟**当且仅当 $(\emptyset, \emptyset) \in R$, 并且如果 $(X, Y) \in R$, 则

- $X \xrightarrow{A}_{\mathcal{E}} X', A \in \mathbb{N}^{\mathrm{Act}} \Rightarrow \exists Y' : Y \xrightarrow{A}_{\mathcal{F}} Y' \wedge (X', Y') \in R$,

- $Y \xrightarrow{A}_{\mathcal{F}} Y', A \in \mathbb{N}^{\mathrm{Act}} \Rightarrow \exists X' : X \xrightarrow{A}_{\mathcal{E}} X' \wedge (X', Y') \in R$.

事件结构 $\mathcal{E}$ 和事件结构 $\mathcal{F}$ 称作**步进互模拟等价**(表示为 $\mathcal{E} \approx_{\mathrm{sb}} \mathcal{F}$)，当且仅当 $\mathcal{E}$ 和 $\mathcal{F}$ 之间存在一个步进互模拟.

与交织等价一样，步进互模拟等价 $\mathcal{E} \approx_{\mathrm{sb}} \mathcal{F}$ 蕴含步进迹等价 $\mathcal{E} \approx_{\mathrm{st}} \mathcal{F}$. 并且，步进互模拟等价和步进迹等价在动作细化下不保持 [41]. 但是，与交织等价一样，如果将事件结构的一些性质作限制，它们在动作细化下也是保持的.

**定理 6.3.1** 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$. 如果事件结构中的事件间的独立关系满足：$\mathrm{co}_{\mathcal{E}} = \mathrm{co}_{\mathcal{F}} = \emptyset$，并且 ref 是一个细化函数，则

(1) $\mathcal{E} \approx_{\mathrm{st}} \mathcal{F} \Rightarrow \mathrm{ref}(\mathcal{E}) \approx_{\mathrm{st}} \mathrm{ref}(\mathcal{F})$,

(2) $\mathcal{E} \approx_{\mathrm{sb}} \mathcal{F} \Rightarrow \mathrm{ref}(\mathcal{E}) \approx_{\mathrm{sb}} \mathrm{ref}(\mathcal{F})$.

**证明** (1) 由于 $\mathrm{co}_{\mathcal{E}} = \mathrm{co}_{\mathcal{F}} = \emptyset$，所以在事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 中的任何一个变迁都是单个动作变迁. 又由于命题 6.3.1, 则有 $\mathcal{E} \approx_{\mathrm{st}} \mathcal{F} \Leftrightarrow \mathcal{E} \approx_{\mathrm{it}} \mathcal{F}$. 根据定理 6.2.3(1), 则有结论 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{it}} \mathrm{ref}(\mathcal{F})$，也就是 $\mathrm{tr}s(\mathrm{ref}(\mathcal{E})) = \mathrm{tr}s(\mathrm{ref}(\mathcal{F}))$. 在事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 中，每个动作标记完全一样，并以相同的方式被细化，同时，因为存在 $\mathrm{co}_{\mathcal{E}} = \mathrm{co}_{\mathcal{F}} = \emptyset$，因此在细化后的事件结构 $\mathrm{ref}(\mathcal{E})$ 和 $\mathrm{ref}(\mathcal{E})$ 中存在相同的动作多重集 (在集合 Act 上) 并且这些动作是并发的，所以可以得出 $\mathrm{steptr}s(\mathrm{ref}(\mathcal{E})) = \mathrm{steptr}s(\mathrm{ref}(\mathcal{F}))$，即 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{st}} \mathrm{ref}(\mathcal{F})$.

(2) 由于 $\mathrm{co}_{\mathcal{E}} = \mathrm{co}_{\mathcal{F}} = \emptyset$，所以在事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 中的任何一个变迁都是单个动作变迁. 因为命题 6.3.1，则有 $\mathcal{E} \approx_{\mathrm{sb}} \mathcal{F} \Leftrightarrow \mathcal{E} \approx_{\mathrm{ib}} \mathcal{F}$. 根据定理 6.2.3(2)，则有 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{F})$. 在事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 中，每个动作标记完全一样，并以相同的方式被细化，同时因为存在 $\mathrm{co}_{\mathcal{E}} = \mathrm{co}_{\mathcal{F}} = \emptyset$，因此在细化后的事件结构 $\mathrm{ref}(\mathcal{E})$ 和 $\mathrm{ref}(\mathcal{E})$ 中存在相同的动作多重集 (在集合 Act 上) 并且这些动作是并发的，同时存在相同的分支时间特性，所以可以得出 $\mathrm{ref}(\mathcal{E}) \approx_{\mathrm{ib}} \mathrm{ref}(\mathcal{F}) \Rightarrow \mathrm{ref}(\mathcal{E}) \approx_{\mathrm{sb}} \mathrm{ref}(\mathcal{F})$.

## 6.4 动作细化下等价的保持

定理 6.3.1 和 定理 6.2.3 表明，如果事件结构中不存在事件间的独立关系，则建立在这些事件结构上的两类交织等价和两类步进等价在动作细化下是保持的，即“没有并发的交织等价”和“没有并发的步进等价”在动作细化下是不变量. 这一节将其中的限制条件放开一点，讨论存在并发情况的的交织等价和步进等价在何种情况下等价的保持问题.

### 6.4.1 束动作变迁

在这一子节里，引入一类新的动作变迁形式：$X \xrightarrow{A} X'$，其中 $A$ 是动作集合 Act 上的多重集合，它里面的动作都是并发执行的，称这个动作多重集合为“一个束动作”，并称这种变迁为“束动作变迁”，束动作变迁实际上是一类特殊的步进动作变迁，当然也包含没有并发的单个动作变迁. 用这种变迁关系，将建立起两类新的等价. 首先给出束动作变迁的定义.

**定义 6.4.1**(束动作变迁) 设事件结构 $\mathcal{E} \in E$. 一个变迁 $X \xrightarrow{A} X'$ 称作一个**束动作变迁**，当且仅当 $A \in \mathbb{N}^{\mathrm{Act}}$(即 $A$ 是动作集合 Act 上的多重集合)，$X, X' \in C(\mathcal{E}), X \subseteq X'$，并且 $X' - X = G$，使得集合 $G$ 中的事件满足：

(1) **完全原因独立关系**：$\forall d, e \in G : (d \ \mathrm{co}_{X'} \ e) \wedge (\{e_1 \in E_{\mathcal{E}} \mid e_1 \ \mathrm{co}_{\mathcal{E}} \ e\} \cup \{e\} = \{e_2 \in E_{\mathcal{E}} \mid e_2 \ \mathrm{co}_{\mathcal{E}} \ d\} \cup \{d\} = G)$ 和 $l_{\mathcal{E}}(G) = A$；

(2) **相同的原因关系**：$\forall e_1 \in E_{\mathcal{E}} \setminus G, \exists e_2 \in G : (e_1 < e_2 \Rightarrow \forall e_3 \in G : e_1 < e_3) \vee (e_1 > e_2 \Rightarrow \forall e_3 \in G : e_1 > e_3)$；

(3) **相同的矛盾关系**：$\forall e_1 \in E_{\mathcal{E}} \setminus G, \exists e_2 \in G : (e_1 \# e_2 \Rightarrow \forall e_3 \in G : e_1 \# e_3)$. 这里，$l_{\mathcal{E}}(G) \in \mathbb{N}^{\mathrm{Act}}$ 由 $l_{\mathcal{E}}(G)(a) = |\{e \in G \mid l_{\mathcal{E}}(e) = a\}|$ 给出.

束动作变迁 $X \xrightarrow{A}_{\mathcal{E}} X'$ 表示在事件结构 $\mathcal{E}$ 中，通过并发执行集合 $A$ 中的所有动作，格局 $X$ 表示的状态可能进化成 $X'$ 表示的状态. 这个束动作变迁关系关联着一个标记束动作变迁系统 (参见第七章的定义 7.3.1) 和一个事件结构. 因此，定义在标记变迁系统上的等价直接诱导出定义在事件结构上对应的等价.

**例 6.4.1** 在一个简单的进程 $P = (a||b); c + f$ 中，它的动作名分别对应事件：$e_a, e_b, e_c$ 和 $e_f$. 假定进程 $P$ 的事件结构模型是 $\mathcal{E}_P$，图 6.1 描述了事件结构 $\mathcal{E}_P$ 的单个动作变迁和格局情况，而图 6.2 则描述了它的束动作变迁和格局分布情况. 从两图可以看出，系统到达格局 $\{e_a, e_b\}$，在图 6.1 中存在五个可能的变迁，而在图 6.2 中仅有一个束动作变迁.

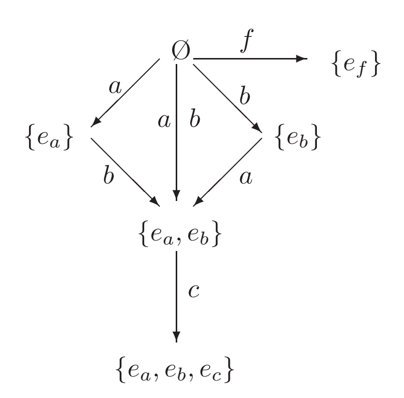



图 6.1 一个事件结构的单个动作变迁

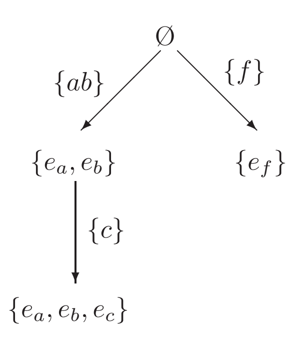



图 6.2　一个事件结构的束动作变迁

从该例可以看出，束动作变迁可能简化单个动作变迁表示的系统，如果将此思想用于简化刻画系统的方法中，可能得到一些好的结果，下一章将详细讨论这个问题.

接下来给出一个新的概念：假设 $e$ 是事件结构 $\mathcal{E}$ 的事件集合中的一个事件，$e$ 的**原因独立关系集** $\theta(e)$(包括自己) 定义为：$\theta(e) = \{e_1 \in E_{\mathcal{E}} | e_1 \text{ co}_{\mathcal{E}}\ e\} \cup \{e\}$.

为研究后面的问题，先给出一个命题.

**命题 6.4.1**　设事件结构 $\mathcal{E} \in E$.

(1) 事件结构 $\mathcal{E}$ 中的每个单个动作变迁都是束动作变迁.

(2) 如果事件结构 $\mathcal{E}$ 中的所有变迁都是束动作变迁，则

① $\forall d \in E_{\mathcal{E}} \Rightarrow \exists(X \xrightarrow{A} X')$：$\theta(d) = G$，其中 $A \in \mathbb{N}^{\text{Act}}$, $X, X' \in C(\mathcal{E}), X \subseteq X'$, $X' - X = G$ 和 $l_{\mathcal{E}}(G) = A$;

② $\forall(X \xrightarrow{A} X') \Rightarrow \exists d \in E_{\mathcal{E}}$：$G = \theta(d)$，其中 $A \in \mathbb{N}^{\text{Act}}$, $X, X' \in C(\mathcal{E}), X \subseteq X'$, $X' - X = G$ 和 $l_{\mathcal{E}}(G) = A$;

③ $\bigcup_{e \in E_{\mathcal{E}}} \theta(e) = E_{\mathcal{E}}$;

④ $\forall e_1, e_2 \in E_{\mathcal{E}}:\ e_1 \neq e_2 \wedge \neg(e_1 \text{ co } e_2) \Rightarrow \theta(e_1) \cap \theta(e_2) = \emptyset$.

**证明**　(1) 该结论很容易证明，此处省略.

(2) ①、②和③的结论很容易得出，此处省略. 只给出④的证明. 设 $\theta(e_1) \cap \theta(e_2) \neq \emptyset$，则 $\exists e \in \theta(e_1) \cap \theta(e_2)$. 根据定义 6.4.1 可得 $\theta(e) = \theta(e_1) = \theta(e_2)$，则有 $e_1 \text{ co } e_2$，或者 $e_1 = e_2$，这与 $e_1 \neq e_2 \wedge \neg(e_1 \text{ co } e_2)$ 矛盾. 因此有 $\forall e_1, e_2 \in E_{\mathcal{E}}:\ e_1 \neq e_2 \wedge \neg(e_1 \text{ co } e_2) \Rightarrow \theta(e_1) \cap \theta(e_2) = \emptyset$.

上面的命题表明：如果一个事件结构的所有变迁都是束动作变迁，则可以将每个动作束变迁中涉及的并发动作看成一个 “大” 动作，相应地，它们对应的事件也可以看成一个 “大” 事件，那么在这种处理下，该事件结构中就没有了事件之间的独立关系. 而且，事件的原因独立关系集合将该事件结构的事件集合划分成不同的部分，如此划分将诱导出该事件结构的事件集合上的一个等价关系.

这里的束动作变迁是一类特殊的步进变迁. 在一个束动作变迁中, 存在完全原因独立关系意味着该动作只能属于一个束动作, 不能被别的束动作共享. 例如, 对于进程 $(a;b)||c$, 由于动作 $c$ 可能属于两个束动作, 即 $\{a,c\}$ 和 $\{b,c\}$, 所以该进程对应的事件结构不满足完全原因独立关系这个条件, 它的变迁不是束动作变迁, 只是一般的步进动作变迁. 这就是为什么进程 $(a;b)||c$ 步进迹等价于进程 $(a||c);b+a;(b||c)$, 但在动作细化后步进迹等价不被保持的真正原因 (参见文献 [41] 中的例 7.1). 又因为并发出现的事件有相同的原因关系和矛盾关系, 所以能将每个束动作看成一个大的动作, 这个性质保证了基于束动作变迁建立的等价在动作细化下是保持的. 因此, 引入基于束动作变迁的等价.

**定义 6.4.2**(束迹) 设事件结构 $\mathcal{E} \in E$. 一个序列 $W = A_1 \cdots A_n (A_i \in \mathbb{N}^{\mathrm{Act}} (i = 1, \cdots, n))$ 称作事件结构 $\mathcal{E}$ 的一个**束迹** 当且仅当 $\exists X_0, \cdots, X_n \in C(\mathcal{E}) : X_0 = \emptyset$ 和 $X_{i-1} \xrightarrow{A_i} X_i (i = 1, \cdots, n)$ 是一个束动作变迁.

本书将用 $Bdl\mathrm{trs}(\mathcal{E})$ 表示事件结构 $\mathcal{E}$ 的所有束迹组成的集合. 接下来给出“束迹等价”的定义.

**定义 6.4.3**(束迹等价) 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$, 并且在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁. $Bdl\mathrm{trs}(\mathcal{E})$, $Bdl\mathrm{trs}(\mathcal{F})$ 分别表示事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 的所有束迹组成的集合. $\mathcal{E}$ 和 $\mathcal{F}$ 称作**束迹等价**(表示为 $\mathcal{E} \approx_{\mathrm{bt}} \mathcal{F}$) 当且仅当 $Bdl\mathrm{trs}(\mathcal{E}) = Bdl\mathrm{trs}(\mathcal{F})$.

下面的命题表明束迹等价与步进迹等价在一定条件下是一致的.

**命题 6.4.2** 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$. 如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁, 则 $\mathcal{E} \approx_{\mathrm{bt}} \mathcal{F} \Leftrightarrow \mathcal{E} \approx_{\mathrm{st}} \mathcal{F}$.

**证明** 根据定义 6.3.1 和 定义 6.4.1, 直接获得上述结论.

下面的命题表明束迹等价蕴含步进迹等价.

**命题 6.4.3** 设两个事件结构 $\mathcal{E}, \mathcal{F} \in E$. 如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁, 则 $\mathcal{E} \approx_{\mathrm{bt}} \mathcal{F} \Rightarrow \mathcal{E} \approx_{\mathrm{it}} \mathcal{F}$.

**证明** 根据定义 6.4.1 并且因为命题 6.4.1, 对于一个束迹中, 任何两个束动作多重集 $A_i, A_j (i = 1, \cdots, n, j = 1, \cdots, n, i \neq j)$, 则 $A_i$ 中的所有动作与 $A_j$ 中的所有动作是独立的. 因此, $A_i$ 中的所有动作与 $A_j$ 中的所有动作执行互不影响. 所以有 $Bdl\mathrm{trs}(\mathcal{E}) = Bdl\mathrm{trs}(\mathcal{F}) \Rightarrow \mathrm{trs}(\mathcal{E}) = \mathrm{trs}(\mathcal{F})$, 即 $\mathcal{E} \approx_{\mathrm{bt}} \mathcal{F} \Rightarrow \mathcal{E} \approx_{\mathrm{it}} \mathcal{F}$.

这里, 对于一个束迹中, 任何两个束动作多重集 $A_i, A_j (i = 1, \cdots, n, j = 1, \cdots, n, i \neq j)$, 则 $A_i$ 中的所有动作与 $A_j$ 中的所有动作是独立的. 因此, $A_i$ 中的所有动作与 $A_j$ 中的所有动作执行互不影响. 因此, 设 $W = A_1 \cdots A_n$, 其中, $A_i \in \mathbb{N}^{\mathrm{Act}} (i = 1, \cdots, n)$ 是一个束迹, 并设 $|A_i|$ 表示多重集合 $A_i$ 中元素的数目, 那么束迹 $W$ 对应着 $|A_1|! \times |A_2|! \times \cdots \times |A_n|!$ 个一般迹. 在例 6.4.1 中, 束迹 $\{a,b\}\{c\}$ 对应着两个 (即 $|\{a,b\}|! \times |\{c\}|! = (2 \times 1) \times (1) = 2$ ) 一般迹 $abc, bac$.

讨论了上述束迹等价后, 讨论另一类新等价, 并研究它们的动作细化下的保持

问题. 这类等价称作“束互模拟等价”，下面给出它的定义.

**定义 6.4.4**(束互模拟等价) 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$, 并假定它们中的所有变迁是束动作变迁. 一个变迁 $R\subseteq C(\mathcal{E})\times C(\mathcal{F})$ 称作事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 的**一个束互模拟**，当且仅当 $(\emptyset,\emptyset)\in R$, 并且如果 $(X,Y)\in R$, 则

- $X\xrightarrow{A}_{\mathcal{E}}X',A\in\mathbb{N}^{\mathrm{Act}}\Rightarrow\exists Y':Y\xrightarrow{A}_{\mathcal{F}}Y'\wedge(X',Y')\in R$,
- $Y\xrightarrow{A}_{\mathcal{F}}Y',A\in\mathbb{N}^{\mathrm{Act}}\Rightarrow\exists X':X\xrightarrow{A}_{\mathcal{E}}X'\wedge(X',Y')\in R$.

$\mathcal{E}$ 和 $\mathcal{F}$ 称作**束互模拟等价**(表示为 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}$)，当且仅当 $\mathcal{E}$ 和 $\mathcal{F}$ 之间存在一个束互模拟.

显然，根据上述定义，有如下结论：$\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}$. 为了更好地研究动作细化下等价的保持问题，先给出一个命题.

**命题 6.4.4** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$. 如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，则 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Leftrightarrow\mathcal{E}\approx_{\mathrm{sb}}\mathcal{F}$.

证明从略.

上述命题表明束互模拟等价 $\approx_{\mathrm{bb}}$ 与步进等价 $\approx_{\mathrm{sb}}$ 在一定条件下是一致的. 接下来，还有以下命题.

**命题 6.4.5** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$. 如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，则 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{ib}}\mathcal{F}$.

证明过程类似于命题 6.4.3 的证明过程，此处省略.

显然这个命题给出了交织语义与步进语义的部分关系.

### 6.4.2 交织等价的保持

文献 [41] 已经证明了在一般情况下交织等价在动作细化下是不保持的，然而，定理 6.2.3 表明没有并发的交织等价在动作细化下是保持的. 这一小节将证明在并发存在的情况下，交织等价在一定条件下在动作细化下保持. 为了证明这个结论，先讨论束迹等价和偏序等价的关系.

**命题 6.4.6** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$. 如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，则 $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{pt}}\mathcal{F}$.

**证明** 对于任意格局 $X\in C(\mathcal{E})$，因为事件结构 $\mathcal{E}$ 中的所有变迁都是束动作变迁，则存在唯一一个束迹，比如假设是 $W$，它的形成开始于初始格局 $\emptyset$ 并进化成格局 $X$. 又由于 $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}$，有 $Bdl\mathrm{trs}(\mathcal{E})=Bdl\mathrm{trs}(\mathcal{F})$，则存在一个格局 $Y\in C(\mathcal{F})$ 是由于执行束迹 $W$ 到达的. 显然，$X\simeq_{\mathrm{p}}Y$，即 $\forall X\in C(\mathcal{E})\Rightarrow\exists Y\in C(\mathcal{F})\wedge X\simeq_{\mathrm{p}}Y$. 利用对称性得出：$\forall Y\in C(\mathcal{F})\Rightarrow\exists X\in C(\mathcal{E})\wedge Y\simeq_{\mathrm{p}}X$. 因为同构保证有相同的标记，所以有 $\mathrm{pomsets}(\mathcal{E})=\mathrm{pomsets}(\mathcal{F})$，即 $\mathcal{E}\approx_{\mathrm{pt}}\mathcal{F}$.

现在讨论交织迹等价在一定条件下的动作细化的保持问题.

**定理 6.4.1** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$. 如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，并且设 ref 是一个细化函数. 如果 $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}$，则 $\mathcal{E}\approx_{\mathrm{it}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{it}}\mathrm{ref}(\mathcal{F})$.

**证明** 因为命题 6.4.6，则有 $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{pt}}\mathcal{F}$. 又因为定理 6.2.1，所以有 $\mathcal{E}\approx_{\mathrm{pt}}\mathcal{F}\Rightarrow\mathrm{ref}(\mathcal{E})\approx_{\mathrm{pt}}\mathrm{ref}(\mathcal{F})$. 而且，因为命题 6.2.1，则有 $\mathrm{ref}(\mathcal{E})\approx_{\mathrm{it}}\mathrm{ref}(\mathcal{F})$. 因此可以得出 $\mathcal{E}\approx_{\mathrm{it}}\mathcal{F}\Rightarrow\mathrm{ref}(\mathcal{E})\approx_{\mathrm{it}}\mathrm{ref}(\mathcal{F})$.

这个定理表明，如果事件结构的所有变迁都是束动作变迁，并且束迹等价成立，则交织等价在动作细化下保持，这就是交织等价在并发存在时动作细化下可保持的重要结论，也是本书讨论所要达到的目标之一. 下面将讨论交织互模拟等价的保持问题.

为方便后面的讨论，先给出一个命题.

**命题 6.4.7** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$. 如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，则 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{h}}\mathcal{F}$.

**证明** 假设关系 $R\subseteq C(\mathcal{E})\times C(\mathcal{F})$ 是两事件结构 $\mathcal{E}$ 和 $\mathcal{F}$ 之间的一个束互模拟. 用归纳法构造出这两个事件结构的一个保持历史互模拟 $\widetilde{R}\subseteq C(\mathcal{E})\times C(\mathcal{F})$: (1) 首先让元素 $(\varnothing,\varnothing)$ 加入集合 $\widetilde{R}$; (2) 现在假定某一元素 $(X,Y)$ 已经是集合 $\widetilde{R}$ 中的一个元素，而且因为 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}$，假定 $X\xrightarrow{A}_{\mathcal{E}}X',Y\xrightarrow{A}_{\mathcal{F}}Y',A\in\mathbb{N}^{\mathrm{Act}}$ 和 $(X,Y)\in R$，又由于 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}$，并由于定义 6.4.2，则有 $\mathcal{E}\approx_{\mathrm{pt}}\mathcal{F}$，因而假定 $X\simeq_{\mathrm{p}}Y$ 和 $X'\simeq_{\mathrm{p}}Y'$. 如果多重动作集合 $A$ 中存在 $n(n>1)$ 个元素，它们交织执行，那么存在 $n!$ 个这样的单个动作变迁序列：$X\xrightarrow{a_1}_{\mathcal{E}}X_1\xrightarrow{a_2}_{\mathcal{E}}X_2\cdots X_{n-1}\xrightarrow{a_{n-1}}_{\mathcal{E}}X'$. 相应地，存在对应的单个动作变迁序列：$Y\xrightarrow{a_1}_{\mathcal{F}}Y_1\xrightarrow{a_2}_{\mathcal{F}}Y_2\cdots Y_{n-1}\xrightarrow{a_{n-1}}_{\mathcal{F}}Y'$，并且 $X_1\simeq_{\mathrm{p}}Y_1$, $X_2\simeq_{\mathrm{p}}Y_2,\cdots,X_{n-1}\simeq_{\mathrm{p}}Y_{n-1}$. 显然，增加这 $n$ 个元素 $(X_i,Y_i)(i=1,\cdots,n)$ 到集合 $\widetilde{R}$，同时增加元素 $(X',Y')$ 到集合 $\widetilde{R}$，这里有 $X_i\simeq_{\mathrm{p}}Y_i$. 沿着这条路线，构造出集合 $\widetilde{R}$ 的所有元素，这些元素是分别从格局 $X$ 和 $Y$ 通过相同的单个动作变迁得到的格局对. 因此 $\widetilde{R}$ 是一个保持历史互模拟，即 $\mathcal{E}\approx_{\mathrm{h}}\mathcal{F}$.

这个命题表明束互模拟等价 $\approx_{\mathrm{bb}}$ 蕴含保持历史等价 $\approx_{\mathrm{h}}$. 接下来讨论束互模拟等价 $\approx_{\mathrm{bb}}$ 与交织互模拟等价在动作细化下的关系.

**定理 6.4.2** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$，设在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，并且设 ref 是一个细化函数. 如果 $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}$，则 $\mathcal{E}\approx_{\mathrm{ib}}\mathcal{F}\Rightarrow\mathrm{ref}(\mathcal{E})\approx_{\mathrm{ib}}\mathrm{ref}(\mathcal{F})$.

**证明** 因为命题 6.4.7，所以有 $\mathcal{E}\approx_{\mathrm{h}}\mathcal{F}$. 又根据定理 6.2.2，则有 $\mathrm{ref}(\mathcal{E})\approx_{\mathrm{h}}\mathrm{ref}(\mathcal{F})$. 而且，根据命题 6.2.2，也有 $\mathrm{ref}(\mathcal{E})\approx_{\mathrm{ib}}\mathrm{ref}(\mathcal{F})$. 因此，可以得出 $\mathcal{E}\approx_{\mathrm{ib}}\mathcal{F}\Rightarrow\mathrm{ref}(\mathcal{E})\approx_{\mathrm{ib}}\mathrm{ref}(\mathcal{F})$.

这个定理表明，如果所有变迁都是束动作变迁，并且存在束互模拟等价，则交织互模拟等价在动作细化下也是保持的. 因此，定理 6.4.1 和定理 6.4.2 扩展了定理 6.2.3，表明了具有并发性质的交织等价在一定条件下也在动作细化下保持.

### 6.4.3 步进等价的保持

文献 [41] 已经证明，在一般情况下，步进等价在动作细化下不保持. 然而，定理 6.3.1 表明没有并发的步进等价在动作细化下是保持的. 由此可见，在一定条件下，步进等价在动作细化下可以保持. 本节将扩展定理 6.3.1，给出具有并发的步进等价的保持问题.

**定理 6.4.3** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$，如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，并且 ref 是一个细化函数，则

(1) $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{st}}\mathrm{ref}(\mathcal{F})$；

(2) $\mathcal{E}\approx_{\mathrm{st}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{st}}\mathrm{ref}(\mathcal{F})$.

**证明** (1) 根据命题 6.4.2 可得 $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{st}}\mathcal{F}$. 根据定义 6.4.1 和命题 6.4.3 可知，一个束动作变迁对应着束动作交织执行形成的多个单个动作变迁，同时单个动作的细化，对应的束动作也被细化，并且可以将每个束变迁中涉及的并发动作看成一个动作，相应地，它们对应的事件也可以看成一个事件，那么在这种处理下，这个事件结构中就没有了原因独立关系. 因此，可以由定理 6.3.1(1) 推导出 $\mathcal{E}\approx_{\mathrm{bt}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{st}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{st}}\mathrm{ref}(\mathcal{F})$.

(2) 根据命题 6.4.2 和上面的推导过程，直接可以得出 $\mathcal{E}\approx_{\mathrm{st}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{st}}\mathrm{ref}(\mathcal{F})$.

这个定理表明，如果所有的变迁都是束动作变迁，步进迹等价在动作细化下是保持的. 下面接着讨论束互模拟等价 $\approx_{\mathrm{bb}}$ 和步进互模拟等价 $\approx_{\mathrm{sb}}$ 在动作细化下的关系，以及步进互模拟的保持问题.

**定理 6.4.4** 设两个事件结构 $\mathcal{E},\mathcal{F}\in E$，如果在 $\mathcal{E}$ 和 $\mathcal{F}$ 中的所有变迁都是束动作变迁，并且 ref 是一个细化函数，则

(1) $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{sb}}\mathrm{ref}(\mathcal{F})$；

(2) $\mathcal{E}\approx_{\mathrm{sb}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{sb}}\mathrm{ref}(\mathcal{F})$.

**证明** (1) 根据命题 6.4.4，可得 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{sb}}\mathcal{F}$. 由命题 6.4.5 和定义 6.4.1 可知，一束动作变迁对应着束动作交织执行形成的多个单个动作变迁，同时单个动作的细化，对应的束动作也被细化，并且可以将每个束变迁中涉及的并发动作看成一个动作，相应地，它们对应的事件也可以看成一个事件，那么在这种处理下，这个事件结构中就没有了原因独立关系. 因此，可以用定理 6.3.1(2) 可以推导出 $\mathcal{E}\approx_{\mathrm{bb}}\mathcal{F}\Rightarrow\mathcal{E}\approx_{\mathrm{sb}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{sb}}\mathrm{ref}(\mathcal{F})$.

(2) 根据命题 6.4.4 和上面 (1) 的结论，直接有 $\mathcal{E}\approx_{\mathrm{sb}}\mathcal{F}\Rightarrow \mathrm{ref}(\mathcal{E})\approx_{\mathrm{sb}}\mathrm{ref}(\mathcal{F})$.

该定理表明如果所有的变迁都是束动作变迁，则步进互模拟等价在动作细化下是保持的.

因此，所有的变迁都是束动作变迁，两类步进等价：步进迹等价和步进互模拟等价在动作细化下保持. 定理 6.4.3 和定理 6.4.4 扩展了定理 6.3.1.

# 6.5 小　结

本章证明了：

(1) 在事件之间没有独立关系的事件结构上的交织迹等价和交织互模拟等价在动作细化下是保持的；

(2) 在事件结构中，如果所有的变迁都是束动作变迁，事件结构之间的束迹等价保证了交织迹等价在动作细化下是保持的；同样，事件结构之间的束互模拟等价也保证了交织互模拟等价在动作细化下是保持的；

(3) 在事件之间没有独立关系的事件结构上的步进迹等价和步进互模拟等价在动作细化下是保持的；

(4) 在事件结构中，如果所有的变迁都是束动作变迁，步进迹等价和步进互模拟等价也在动作细化下保持.

因此，我们找到了一类具有特定性质的并发进程，它们能使交织等价和步进等价在没有限制的动作细化的条件下是保持的.

在这里，我们提出了“束动作变迁”的概念，这种处理系统变迁的方式不同于基于迹理论采用“偏序约简”以减少并发系统的状态，而是将完全并发的动作看成一个“大动作”，使得系统在这个大动作执行的前后仅存在两个状态，见例 6.4.1. 由此，束动作变迁的概念能够用于处理系统验证过程中出现的状态爆炸问题. 这些工作进一步完善了动作细化理论，解决了交织等价和步进等价在动作细化下的保持问题，并将其中得到的部分成果用于偏序约简方法，取得了实际应用的支持，为进一步研究拓展了思路. 下一章我们将束动作变迁思想引入到偏序约简 [20]，改造偏序约简方法，同时讨论如何应用于模型检验 [23].

# 第七章　基于束动作的偏序约简

## 7.1　引　　言

在形式化验证技术当中，模型检验作为一种机械化的验证方法，在学术界得到了广泛注意，并在工业上受到了普遍重视. 模型检验的基本思想是用时态逻辑公式表达系统所期望的性质，用有限状态机 (finite state machine，FSM) 表示系统的状态转移结构，通过遍历 FSM 来检验时态逻辑公式的正确性. 如果不能验证公式的正确性，系统将根据遍历的路径给出一个反例，使得人们发现公式不成立的原因，帮助人们改正错误. 随着软件和硬件等并发系统规模的增大，状态数目呈指数增加而引起组合爆炸，这就限制了模型检验的应用. 20 世纪 80 年代， Bryant 提出了用二叉判定图 BDD(binary decision diagram) 表示布尔函数，并以 BDD 作为在计算机中的存储方式，极大地减少了存储布尔公式所需要的空间，在一定程度上控制了状态爆炸问题. 但是这种方法仍然不能有效解决状态爆炸问题. 20 世纪 90 年代，人们提出了符号模型检验 (symbolic model checking)[16] 的方法，并与 BDD 结合，进一步控制了状态空间的爆炸. 与此同时，出现了抽象技术，通过不加解释的符号可以得到系统的抽象，抽象隐藏了与验证无关的细节，因此抽象后的系统比较简单，但保留了验证所需要的足够信息，降低了系统验证的复杂度，例如多路决策图 MDG(multiway decision graphs)[19]. 另一种有效的方法是定义等价关系，通过合并系统中的等价状态来减少系统的状态数目，偏序约简 (partial order reduction)[4,22,43,44,75,87] 就属于这类方法，但它有自己的特点.

偏序约简的基本思想，是在迹理论 (trace theory)[67] 的基础上建立动作独立等价关系，利用动作独立关系和路径扫描迹等价 (stuttering equivalence) 关系建立起执行动作序列的等价关系，对状态空间进行有选择的访问，达到减小访问状态空间的目的. 这种方法最基础的部分是对动作的独立关系和路径扫描迹等价的确定. 一般认为，在并发异步进程中，并发动作相互独立，如果它们交织执行最终到达相同的状态，就认为它们交织执行是等价的. 显然，只要首先确定动作是否相互并发执行，其次 (如果独立) 确定它们交织执行是否从相同的状态出发最终又到达相同的状态，最后才说它们的执行是等价的，一般称为“动作独立”. 图 7.1 给出了状态变迁关系下的两个动作独立. 但是单个动作间的独立等价关系，不但粒度较小，而且无法解决图 7.2 与图 7.3 所示的问题.

偏序约简主要是针对并发异步进程，在图 7.2 中，动作 $\beta_1$ 和 $\beta_2$ 是真并发 (并行) 执行，与动作 $\alpha$ 可异步并发 (可交织) 执行，若需要利用上述基于动作独立等价进行约简，只有把动作 $\beta_1$ 和 $\beta_2$ 捆绑在一起变成束动作 (看成一个动作) 才能解

决这个问题. 图 7.3 中, 若利用上述基于动作的独立等价进行约简无法进行, 因为四个执行的动作序列是 $\alpha_1\beta_1$, $\alpha_2\beta_2$, $\beta_1\alpha_2$ 和 $\beta_2\alpha_1$. 因此在本书中, 引入了束动作概念, 将这些问题考虑进去. 我们的方法包含基于动作的传统的偏序约简方法. 在过去的十多年中, 许多研究者对传统的偏序约简算法在模型检验中的应用做了大量的工作, 其中具有代表性的有 Valmari[87], Godefroid 与 Wolper[43,44] 和 Peled[75] 等人. 对传统的偏序约简的改进, 一般从两方面进行, 一方面是对算法本身加以改进, 主要方法是加上一些启发式规则 [22], 使模型检验能顺利进行; 另一方面是与其他技术相结合进行改进, 例如与 on-the-fly 模型检验 [75] 结合, 与符号模型检验 [4] 结合.

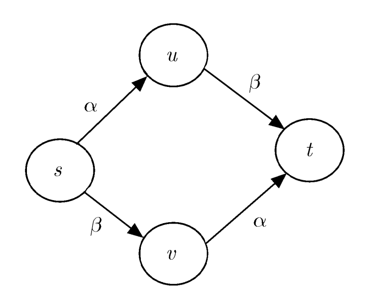



图 7.1　两个动作独立

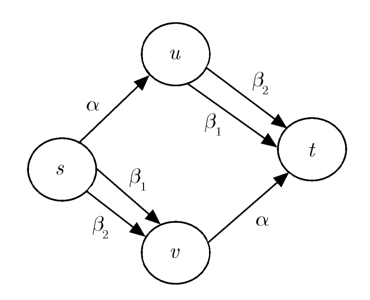



图 7.2　一种简单的独立束动作

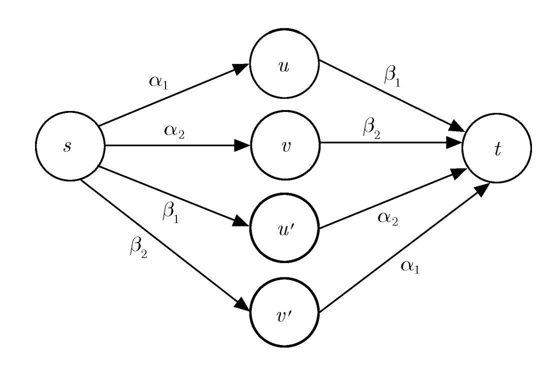



图 7.3　一般的独立束动作

这些改进都停留在对动作粒度上的改进，没有提出比动作粒度更大的束动作概念.

本章的结构是，7.2 节从概念上回顾了基于束动作的偏序约简方法；7.3 节给出了动作与束动作的关系，同时给出了束动作的描述性定义；7.4 节讨论在偏序约简中束动作的基本思想，为了便于叙述，首先建立了基于束动作的标记变迁系统，接着提出了束动作独立和束动作序列独立的概念，最后说明了束动作序列的独立关系；7.5 节介绍了束动作路径扫描迹等价；7.6 节给出了在束动作标记变迁系统上基于束动作的偏序约简的原理；7.7 节介绍了束动作偏序约简的实现方法，给出了从传统的标记变迁系统导出束动作标记变迁系统的具体步骤，并证明了保证这种实现方法的正确性；7.8 节是小结.

## 7.2 传统的偏序约简

### 7.2.1 Kripke 结构

偏序约简一般针对异步并发进程，对异步并发进程的建模一般可以采用 Kripke 结构，Kripke 结构是为模型检验建立的一类特殊的标记变迁系统，下面给出它的定义.

**定义 7.2.1(** Kripke 结构**)** 一个 **Kripke 结构**是一个五元组 $\langle S,\ S_0,\ A,\ \rightarrow,\ L\rangle$，其中，

(1) $S$ 是一个有限的状态集合；

(2) $S_0$ 是初始状态的集合，并且 $S_0 \subseteq S$ ；

(3) $A$ 是动作的集合；

(4) $\rightarrow$ 是变迁关系，并且 $\rightarrow \subset S \times A \times S$ 满足：如果 $\alpha \in A,\ \exists\ s,\ t,\ t'$ : $(s,\ \alpha,\ t) \in \rightarrow$, $(s,\ \alpha,\ t') \in \rightarrow$，那么 $t = t'$；

(5) $L : S \rightarrow p(AP)$ 是标记函数，其中 $AP$ 是原子命题集合，函数 $L$ 是把状态映射为原子命题为真的命题集合，$p(AP)$ 是原子命题幂集.

有时表示系统从状态 $s$ 执行动作 $a$ 后变迁到状态 $t$ 可以写作 $(s, a, t)$ 或 $s \xrightarrow{a} t$，另外可以用后继函数 succ 表示系统从某状态执行一个动作后的后继状态，即 $t = \mathrm{succ}(s, a)$. 把从某状态 $s$ 出发能执行的动作集定义为 $\mathrm{enabled}(s) = \{a \mid \exists t : s \xrightarrow{a} t\}$. 系统的一个执行序列是由动作序列构成，如果动作序列 $\overline{a} = a_1 a_2 \cdots a_n (n > 0)$ 表示该 Kripke 结构中存在执行序列 $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} t$，可以简写成 $s_0 \overset{\overline{a}}{\Longrightarrow} t$，也可以用后继函数表示 $t = \mathrm{succ}(s_0, \overline{a})$. 有时候称 $\overline{a}$ 是该 Kripke 结构的一个**迹**. $|\overline{a}|$ 表示动作序列 $\overline{a}$ 所有动作的集合，即 $|\overline{a}| = \{a_1, a_2, \cdots, a_n\}$. 本书中 $A^*$ 表示在一个标记 Kripke 结构中基于动作集 $A$ 的所有非空的动作序列.

### 7.2.2 动作独立

**定义 7.2.2**(动作独立) 在 Kripke 结构 $\langle S, S_0, A, \to, L\rangle$ 中, 设 $\alpha, \beta \in A(\alpha \neq \beta)$, 对 $\forall s \in S : \alpha, \beta \in \text{enabled}(s)$, 如果动作 $\alpha, \beta$ 满足下面两个条件:

(1) $\exists x, y \in S : x = \text{succ}(\text{succ}(s, \alpha), \beta), y = \text{succ}(\text{succ}(s, \beta), \alpha)$, 并且

(2) $x = y$,

那么称这两个动作 $\alpha$ 和 $\beta$ **相互独立**(如图 7.1), 记为 $\alpha \sim \beta$.

注意独立的动作间不存在相互影响执行的情况. 显然, 对每个 Kripke 结构, 这种建立在动作集上的独立关系是反自反的和对称的, 如果两个动作相互独立, 那么它们执行的先后顺序没有关系, 只与它们所在的起始状态和终止状态有关, 具有一种特定的行为上的对称性. 这种独立等价关系, 可以扩展到动作序列.

**定义 7.2.3**(动作序列独立) 在 Kripke 结构 $\langle S, S_0, A, \to, L\rangle$ 中, 设 $\overline{\gamma_1}, \overline{\gamma_2}, \overline{\delta_1}, \overline{\delta_2} \in A^*$, 并设 $\alpha, \beta \in A$: $\alpha \sim \beta$, 如果 $\exists \overline{\gamma}, \overline{\delta} \in A^* : \overline{\gamma} = \overline{\gamma_1}\alpha\beta\overline{\gamma_2}, \overline{\delta} = \overline{\delta_1}\beta\alpha\overline{\delta_2}$ , 并且 $\overline{\gamma_1} = \overline{\delta_1}, \overline{\gamma_2} = \overline{\delta_2}$, 则称**动作序列 $\overline{\gamma}$ 与 $\overline{\delta}$ 相互独立**, 记为 $\overline{\gamma} = \overline{\delta}$.

根据上述定义, 很容易得到下面的定理.

**定理 7.2.1** 在一个 Kripke 结构中, 建立在动作序列集合 $A^*$ 上, 关于动作序列相互独立的关系是等价关系.

### 7.2.3 扫描迹等价 (stuttering equivalence)

**定义 7.2.4**(路径扫描迹等价) 在 Kripke 结构 $\langle S, S_0, A, \to, L\rangle$ 中, 设两条无限路径 $\sigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots$ 和 $\rho = r_0 \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} \cdots$ , 如果存在两个无限整数序列 $0 = i_0 < i_1 < i_2 < \cdots$ 和 $0 = j_0 < j_1 < j_2 < \cdots$, 使得对任何非负整数 $k$, 有 $L(s_{i_k}) = L(s_{i_k+1}) = \cdots = L(s_{i_{k+1}-1}) = L(r_{j_k}) = L(r_{j_k+1}) = \cdots = L(r_{j_{k+1}-1})$ 成立, 其中 $i_k$ 和 $j_k$ 分别是两条路径中相同的标记 (原子命题) 的状态开始点的下标, 则称**两条路径 $\sigma$ 和 $\rho$ 扫描迹等价**(如图7.4), 记为 $\sigma \approx_{\text{st}} \rho$.

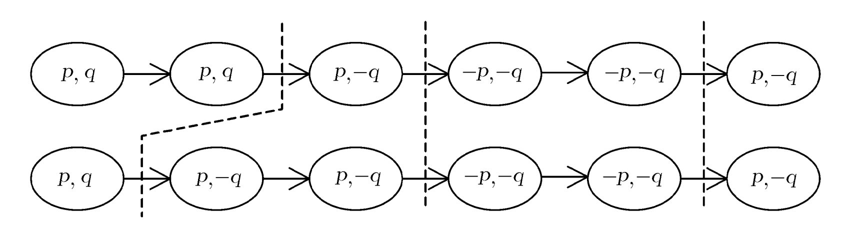



图 7.4 扫描迹等价路径

给定一个 Kripke 结构 $M = \langle S, S_0, A, \to, L\rangle$ 和一个 LTL 公式 $\varphi$, 模型检验就是对每一个初始状态 $s_0(s_0 \in S_0)$, 并且对每一条从 $s_0$ 开始的路径 $\omega$, 验证 $\omega \models \varphi$ 是否成立. 如果都成立, 记为 $M \models \varphi$.

如果有一个 LTL 公式 $\varphi$，两条路径 $\sigma$ 和 $\sigma'$，满足 $\sigma \approx_{\mathrm{st}} \sigma'$，并有 $\sigma \models \varphi$ 当且仅当 $\sigma' \models \varphi$，称这个 LTL 公式 $\varphi$ 是**在扫描迹下的不变量**( an invariant under stuttering). 一般来说，LTL 公式对时序运算符 $X$(neXt) 敏感，为了便于使用路径扫描迹等价关系约简状态空间，表示系统性质的 LTL 公式可以不用时序运算符 $X$(neXt)，用 LTL_x 表示没有时序运算符 $X$ (neXt) 的 LTL 公式集，因为 LTL_x 公式在路径扫描迹等价关系下可以表示所有 LTL 公式 [76].

扫描迹等价的思想可以从路径扩展到 Kripke 结构上.

**定义 7.2.5** 设两个 Kripke 结构 $M$ 和 $M'$，如果 (1) 对 $M$ 中从初始状态开始的每条路径 $\sigma$，在 $M'$ 中也存在一条从初始状态开始的路径 $\sigma'$，并满足 $\sigma \approx_{\mathrm{st}} \sigma'$；(2) 对 $M'$ 中从初始状态开始的每条路径 $\sigma'$，在 $M$ 中也存在一条从初始状态开始的路径 $\sigma$ ，并满足 $\sigma' \approx_{\mathrm{st}} \sigma$；则称**两个 Kripke 结构 $M$ 和 $M'$ 扫描迹等价**.

用 LTL_x 公式表示系统属性，这些公式是在扫描迹等价下的不变量，从定义 7.2.5 可以直接得出定理 7.2.2.

**定理 7.2.2** 如果有两个 Kripke 结构 $M$ 和 $M'$ 扫描迹等价，则对任何表示系统属性的 LTL_x 公式 $\varphi$ , 有

$$M' \models \varphi \ \text{当且仅当}\ M \models \varphi.$$

### 7.2.4 偏序约简

设 $L : S \to p(\mathrm{AP})$ 是定义 7.2.1 中的标记函数，其中 $S$ 是状态集，AP 是原子命题集合，$p(\mathrm{AP})$ 是原子命题幂集. 该函数的作用是用原子命题为真的命题集合来标记系统的每个状态.

**定义 7.2.6**(不可见) 在 Kripke 结构中, 如果执行一个动作 $\alpha$ 导致一个变迁, 并存在一个原子命题集合 $\mathrm{AP}' \subseteq \mathrm{AP}$, 对于每一对状态 $s, s'$ , 使得 $s' = \mathrm{succ}(s, \alpha), L(s) \cap \mathrm{AP}' = L(s') \cap \mathrm{AP}'$, 则称变迁或动作 $\alpha$ **关于原子命题集合 $\mathrm{AP}'$ 不可见**.

如果一个动作或一个变迁不是不可见的，则称它为**可见**的.

偏序约简就是利用动作或变迁不可见与动作相互独立，并在模型检验的算法中实现有选择的遍历状态图, 使访问的模型状态的数目大大减少, 达到减小状态空间的目的. 如果设 Kripke 结构 $M_i$ 是未约简前的全状态图，$M_r$ 是约简后的状态图，其基本思想可以归纳如下：

(1) Kripke 结构 $M_i$ 中从初始状态开始的路径上没有两个动作序列存在独立关系, 因为每条路径上的独立动作序列只保留了一条动作序列, 其他动作序列是不可见的.

(2) 两个 Kripke 结构 $M_i$ 和 $M_r$ 中如果存在路径扫描迹等价，那么 $M_r$ 中路径中没有两个状态间存在不可见的动作或变迁.

## 7.3 动作与束动作

在文献 [4, 22, 43, 44, 75, 87] 中，对动作独立性的一般认识是，如果两个动作相互独立，无论执行的先后都不影响系统执行的结果 (到达相同的状态). 正如图 7.1 所示两个动作 $\alpha$ 和 $\beta$, 都从状态 $s$ 出发, 先执行动作 $\alpha$ 到达状态 $u$, 再执行动作 $\beta$ 到达状态 $t$; 与先执行动作 $\beta$ 到达状态 $v$, 再执行动作 $\alpha$ 到达状态 $t$ 结果都一样, 即从状态 $s$ 变迁到了状态 $t$. 这种处理方式对用传统的 Kripke 结构来刻画并发异步进程是比较好的方案, 但是, 从图 7.2、图 7.3 表示的部分并发异步系统来看, 上面采用以动作为基础的约简方法就不能进行. 在图 7.2 中, 如果将同时执行的动作 $\beta_1$ 和 $\beta_2$ 看成束动作 $\{\beta_1, \beta_2\}$(相当于把它看成一个较大的动作), 这样 $\{\beta_1, \beta_2\}$ 与动作 $\alpha$ (注意, 为了统一, 这里的单动作可以写作 $\{\alpha\}$) 就相互独立了. 在图 7.3 中, 上面采用以动作为基础的约简方法也不能建立独立关系, 因为四个动作 $\alpha_1$, $\alpha_2$, $\beta_1$ 和 $\beta_2$ 能并发执行, 如果它们交织执行, 将走不同的执行路径: $s\alpha_1u\beta_1t$, $s\alpha_2v\beta_2t$, $s\beta_1u'\alpha_2t$ 和 $s\alpha_2v'\beta_1t$. 如果采用束动作, 可以将动作 $\alpha_1$ 和 $\alpha_2$ 看成束 $\{\alpha_1, \alpha_2\}$, 将动作 $\beta_1$ 和 $\beta_2$ 看成束 $\{\beta_1, \beta_2\}$, 这样的束动作 $\{\alpha_1, \alpha_2\}$ 与 $\{\beta_1, \beta_2\}$ 就是相互独立的. 为此, 在这里对束动作作简单的描述性定义 (更精确的形式化定义见定义 7.7.1).

**定义 7.3.1**(束动作)　在一个变迁系统中, **束动作**是单个或多个并发 (一般为真并发) 动作组成的集合.

**性质 1**　束动作是非空的动作组成的集合.

**性质 2**　单个动作可以形成束动作.

在一个 Kripke 结构中, 有 $n(n \geqslant 1)$ 个动作从某个状态出发并发执行后到达 $m(m \geqslant 1)$ 个不同的状态, 并且如果这 $n$ 个动作又从 $m'(m' \geqslant 1)$ 个不同状态并发执行又汇合到另一个状态, 这 $n$ 个动作就可能构成一个束动作. 由此可见, 对于可以并发执行的动作组成的集合, 如果可以看成一个大动作 (相当于执行一个动作), 这个动作构成的集合称为束动作. 有了束动作的定义, 在这里就可以粗略说明相互独立的束动作. 在一个 Kripke 结构中, 如果有一组 (一个) 并发执行的动作 (这些动作组成的集合记为 $X$), 与另外一组 (或一个) 并发执行的动作 (这些动作组成的集合记为 $Y$) 可以异步执行, 可以称集合 $X$ 和 $Y$ 是相互独立的束动作.

## 7.4 束动作的基本思想

为了以束动作为最小粒度叙述基于束动作的偏序约简的基本思想, 首先对并发异步进程建模的 Kripke 结构 (定义 7.2.1) 更改为"束动作 Kripke 结构", 并给出类似于基于动作的偏序约简的原理, 最后再讨论如何将传统的 Kripke 结构 (定义 7.2.1) 更改为束动作 Kripke 结构. 下面给出束动作 Kripke 结构的定义.

**定义 7.4.1**(束动作 Kripke 结构) 一个**束动作 Kripke 结构**是一个五元组 $\langle S,\ S_0,\ A,\ \mapsto,\ L\rangle$，其中，

(1) $S$ 是状态的一个有限集合；

(2) $S_0$ 是初始状态的集合，并且 $S_0 \subseteq S$；

(3) $A$ 是动作的集合；

(4) $\mapsto \subseteq S \times \mathbb{N}^A \times S$ 是变迁关系, 满足：$\mathbb{N}^A$ 是 $A$ 上的多重集合组成的集合 (即束动作集)，并且 $\forall X \in \mathbb{N}^A, \exists s, t, t' : (s, X, t) \in \mapsto, (s, X, t') \in \mapsto \Rightarrow t = t'$；

(5) $L : S \to p(\mathrm{AP})$ 是标记函数，其中 AP 是原子命题集合，函数 $L$ 是把状态映射为原子命题为真的命题集合，$p(\mathrm{AP})$ 是原子命题幂集.

注意，有时表示系统从状态 $s$ 执行束动作 $X$ 后变迁到状态 $t$ 为 $(s, X, t)$ 或 $s \stackrel{X}{\longmapsto} t$，另外可以用后继函数 succ 表示系统从某状态执行一个束动作后的后继状态，即 $t = \mathrm{succ}(s, X)$. 把从某状态 $s$ 出发能执行的束动作集定义为 $\mathrm{enabled}(s) = \{X \mid \exists t : s \stackrel{X}{\longmapsto} t\}$. 系统的一个执行序列是由束动作序列构成，如果束动作序列 $\overline{X} = X_1X_2\cdots X_n(n > 0)$ 表示该 Kripke 结构中存在执行序列 $s_0 \stackrel{X_1}{\longmapsto} s_1 \stackrel{X_2}{\longmapsto} s_2 \cdots \stackrel{X_n}{\longmapsto} t$，可以简写成 $s_0 \stackrel{\overline{X}}{\longmapsto} t$，也可以用后继函数表示 $t = \mathrm{succ}(s_0, \overline{X})$. 有时候称 $\overline{X}$ 是该 Kripke 结构的一个**迹**. $|\overline{X}|$ 表示束动作序列 $\overline{a}$ 所有束动作的集合，即 $|\overline{X}| = \{X_1, X_2, \cdots, X_n\}$. 本书中 $\mathbb{N}^{A^*}$ 表示在一个 Kripke 结构中基于束动作集 $\mathbb{N}^A$ 的所有非空的动作序列.

接下来给出束动作独立和束动作序列独立的概念.

**定义 7.4.2**(束动作独立) 在束动作 Kripke 结构中, 设 $\mathbb{N}^A, S$ 分别是束动作集、状态集，设 $X, Y \in \mathbb{N}^A (X \neq Y)$，对 $\forall s \in S : X, Y \in \mathrm{enabled}(s)$，如果束动作 $X, Y$ 满足下面两个条件：

(1) $\exists s_x, s_y \in S : s_x = \mathrm{succ}(\mathrm{succ}(s, X), Y), s_y = \mathrm{succ}(\mathrm{succ}(s, Y), X)$，并且

(2) $s_x = s_y$，

则称这**两个束动作** $X, Y$ **相互独立**，记为 $X \sim_{\mathrm{ab}} Y$.

注意独立的束动作间不存在相互影响执行的情况.

**定义 7.4.3**(束动作序列独立) 设 $\overline{C_1}, \overline{C_2}, \overline{D_1}, \overline{D_2} \in \mathbb{N}^{A^*}$，并设 $X, Y \in \mathbb{N}^A : X \sim_{\mathrm{ab}} Y$，如果存在 $\overline{C}, \overline{D} \in \mathbb{N}^{A^*} : \overline{C} = \overline{C_1}XY\overline{C_2}, \overline{D} = \overline{D_1}YX\overline{D_2}$，并且 $\overline{C_1} = \overline{D_1}, \overline{C_2} = \overline{D_2}$，称**束动作序列** $\overline{C}$ **与** $\overline{D}$ **相互独立**，记为 $\overline{C} \doteqdot_{\mathrm{ab}} \overline{D}$.

两个相互独立的束动作可视为在某种程度上是行为对称的. 有了这些定义，可以得出如下定理.

**定理 7.4.1** 在束动作集合上，关于束动作序列相互独立的关系是等价关系.

**证明** 设任意 $\overline{C}, \overline{D}, \overline{F} \in \mathbb{N}^{A^*}$.

(1) 因为 $\overline{C} \doteqdot_{\text{ab}} \overline{C}$ 成立，所以上述关系具有自反性；

(2) 因为 $\overline{C} \doteqdot_{\text{ab}} \overline{D}$，根据定义 7.4.3，显然 $\overline{D} \doteqdot_{\text{ab}} \overline{C}$ 成立，所以上述关系具有对称性；

(3) 如果 $\overline{C} \doteqdot_{\text{ab}} \overline{D}$，$\overline{D} \doteqdot_{\text{ab}} \overline{F}$ 成立，则根据定义 7.4.3，有 $|\overline{C}| = |\overline{D}|$，$|\overline{D}| = |\overline{F}|$，那么有 $|\overline{C}| = |\overline{F}|$. 因此，可以设 $\overline{C_1}, \overline{C_2}, \overline{D_1}, \overline{D_2} \in \mathbb{N}^{A^*}$，并设 $X, Y \in \mathbb{N}^A : X \sim_{\text{ab}} Y$，则存在 $\overline{C}, \overline{D} \in \mathbb{N}^{A^*} : \overline{C} = \overline{C_1}XY\overline{C_2}, \overline{D} = \overline{D_1}YX\overline{D_2}$，并且 $\overline{C_1} = \overline{D_1}, \overline{C_2} = \overline{D_2}$. 因为 $\overline{D} \doteqdot_{\text{ab}} \overline{F}$，所以可设 $U, V \in |\overline{C}|$，并且有 $U \sim_{\text{ab}} V$；不失一般性，可设 $U, V \in |\overline{D_2}|$，$\overline{D} = \overline{D_1}YX\overline{D_{21}}UV\overline{D_{22}}$，其中 $\overline{D_2} = \overline{D_{21}}UV\overline{D_{22}}$，$\overline{F} = \overline{F_1}VU\overline{F_2}$，则有 $\overline{F_1} = \overline{D_1}YX\overline{D_{21}} = \overline{C_1}YX\overline{D_{21}}, \overline{F_2} = \overline{D_{22}}$，由上面可以得出 $\overline{C} = \overline{C_1}XY\overline{D_{21}}UV\overline{D_{22}}$ 和 $\overline{F} = \overline{C_1}YX\overline{D_{21}}VU\overline{D_{22}}$，这两等式中，存在两对独立的束动作 $X \sim_{\text{ab}} Y$ 和 $U \sim_{\text{ab}} V$，根据定义 7.4.2，这两个束动作序列 $\overline{C}$ 和 $\overline{F}$ 只有经过两个不同的状态，最终会到达相同的状态，所以有 $\overline{C} \doteqdot_{\text{ab}} \overline{F}$. 因此，上述关系具有传递性.

由上述证明可知，束动作序列相互独立关系是一种等价关系.

## 7.5 束动作路径扫描迹等价

为了便于叙述，下面把束动作导致进程变迁形成的执行路径简称“束动作路径”，如果路径是无限的，简称“束动作无限路径”. 束动作路径是进程执行束动作形成的，是状态和束动作交替形成的路径，它与一般由执行动作形成的路径几乎一致，所以在束动作 Kripke 结构中也存在束动作路径扫描迹等价关系.

**定义 7.5.1**(束动作路径扫描迹等价) 在束动作 Kripke 结构 $\langle S, S_0, A, \to, L\rangle$ 中，设两条无限束动作路径 $\sigma_{\text{ab}} = s_0 \xmapsto{X_1} s_1 \xmapsto{X_2} \cdots$ 和 $\rho_{\text{ab}} = r_0 \xmapsto{X_1} r_1 \xmapsto{X_2} \cdots$，如果存在两个无限整数序列 $0 = i_0 < i_1 < i_2 < \cdots$ 和 $0 = j_0 < j_1 < j_2 < \cdots$，使得对任何非负整数 $k$，有 $L(s_{i_k}) = L(s_{i_k+1}) = \cdots = L(s_{i_{k+1}-1}) = L(r_{j_k}) = L(r_{j_k+1}) = \cdots = L(r_{j_{k+1}-1})$ 成立，其中 $i_k$ 和 $j_k$ 分别是两条路径中相同的标记 (原子命题) 的状态开始点的下标，则称**两条束动作路径 $\sigma_{\text{ab}}$ 和 $\rho_{\text{ab}}$ 扫描迹等价**(如图7.4)，记为 $\sigma_{\text{ab}} \approx_{\text{abst}} \rho_{\text{ab}}$.

束动作路径扫描迹等价的思想仍然可以从路径扩展到整个束动作 Kripke 结构.

**定义 7.5.2** 设两个束动作 Kripke 结构 $M_{\text{ab}}$ 和 $M'_{\text{ab}}$，如果 (1) 对 $M_{\text{ab}}$ 中从初

始状态开始的每条路径 $\sigma_{\rm ab}$，在 $M'_{\rm ab}$ 中也存在一条从初始状态开始的路径 $\sigma'_{\rm ab}$ 满足 $\sigma_{\rm ab} \approx_{\rm abst} \rho_{\rm ab}$；(2) 对 $M'_{\rm ab}$ 中从初始状态开始的每条路径 $\sigma'_{\rm ab}$，在 $M_{\rm ab}$ 中也存在一条从初始状态开始的路径 $\sigma_{\rm ab}$ 满足 $\sigma_{\rm ab} \approx_{\rm abst} \rho_{\rm ab}$. 则称**两个束动作 Kripke 结构 $M_{\rm ab}$ 和 $M'_{\rm ab}$ 扫描迹等价**.

仍然用 LTL_x 公式表示进程属性，这些公式是在扫描迹等价下不变的，由定义 7.5.2 和第二章关于模型检验的基本原理，以及定理 7.2.2，可以直接得到定理 7.5.1.

**定理 7.5.1** 如果有两个束动作 Kripke 结构 $M_{\rm ab}$ 和 $M'_{\rm ab}$ 扫描迹等价，则对任何表示系统属性的 LTL_x 公式 $\varphi$，有 $M_{\rm ab} \models \varphi$ 当且仅当 $M'_{\rm ab} \models \varphi$.

## 7.6 束动作偏序约简

由于我们定义的束动作 Kripke 结构 (定义 7.4.1) 与 Kripke 结构 (定义 7.2.1) 思想相同，因此束动作偏序约简的原理与一般的偏序约简类似.

根据定义 7.4.1 中束动作 Kripke 结构的标记函数 $L : S \to p(\mathrm{AP})$，其中 $S$ 是状态集，AP 是原子命题集合，$P(\mathrm{AP})$ 是原子命题幂集. 该函数的作用是用原子命题为真的命题集合来标记系统每个状态.

**定义 7.6.1**(束动作不可见) 在束动作 Kripke 结构中，如果有一个束动作 $X$，并存在一个原子命题集合 $\mathrm{AP}' \subseteq \mathrm{AP}$，对于每一对状态 $s, s'$，使得 $s' = \mathrm{succ}(s, X)$，$L(s) \cap \mathrm{AP}' = L(s') \cap \mathrm{AP}'$，则称 **束动作 $X$ 关于原子命题集合 AP' 是不可见**的.

如果一个束动作或一个变迁不是不可见的，则称它为**可见**的.

根据模型检验的基本原理，如果两个状态中包含的原子命题一样，并且这两个状态是由一个变迁连在一起的，则可以将两个状态合并，即去掉连在两个状态上的边，将它们收缩为一个状态，这样不会影响验证结果，但状态空间因此而减少.

在并发异步进程中，束动作 Kripke 结构模型在相应的状态图上有大量的状态和路径，例如，有 $n$ 个相互独立的束动作，则有 $n!$ 种排列，相应的状态则有 $2^n$ 个状态，如果选择独立动作序列中具有代表性的一条动作序列，其他 $n!-1$ 种序列在模型检验中都不用考虑，也能完成模型检验，那么状态空间将会减少更多. 目前的模型检验就是假定 $n!-1$ 种执行序列中的状态被合并 ( $n$ 个束动作执行序列对应的路径是扫描迹等价的)，其束动作是不可见的，由此构建成算法实现模型检验. 这样将全状态空间缩减为简化的部分状态空间的过程就是束动作偏序约简的原理. 如果设束动作 Kripke 结构 $M_{\rm iab}$ 是未约简前的全状态图，$M_{\rm rab}$ 是约简后的状态图，

基于束动作的偏序约简的原理可以归纳如下：

(1) 每个变迁是由相应束动作导致的；

(2) 束动作 Kripke 结构 $M_{\rm rab}$ 中，从初始状态开始的路径上没有两个动作序列存在独立关系, 因为每条路径上的独立动作序列只保留了一条动作序列，其他动作序列是不可见的；

(3) 在束动作 Kripke 结构 $M_{\rm rab}$ 中，从初始状态开始的每条路径 $\sigma'_{\rm ab}$，在束动作 Kripke 结构 $M_{\rm iab}$ 中都存在有路径 $\sigma_{\rm ab}$ 与其路径扫描迹等价，即 $\sigma'_{\rm ab} \approx_{\rm abst} \sigma_{\rm ab}$；

(4) 在 $M_{\rm rab}$ 中，每条路径中没有两个状态间存在不可见的束动作或变迁. 显然， $M_{\rm rab}$ 的状态空间比 $M_{\rm iab}$ 的状态空间小很多，起到了减小状态空间的作用.

## 7.7 束动作偏序约简的实现

束动作偏序约简的实现方法可以从进程模型和改造现有的算法两方面考虑. 从进程模型方面来考虑，就是将现有的 Kripke 结构转化为束动作 Kripke 结构，这样处理可以直接采用原有的算法，本书主要采用这种方法. 由于现有的算法涉及到许多启发式规则，改造现有算法实现起来比较困难，下面从进程模型方面考虑.

前面的叙述已经提到对并发异步进程的形式建模采用了两种模型，传统的偏序约简方法一般采用 Kripke 结构，而我们采用了束动作 Kripke 结构，从直观上可以看出，这两种模型存在着差异，例如，从图 7.3 中就可以看出，状态变迁图对应着传统的 Kripke 结构，如果采用束动作 Kripke 结构，则需要将其中的状态 $u$ 与 $v$，$u'$ 与 $v'$ 分别合并，如图 7.5.

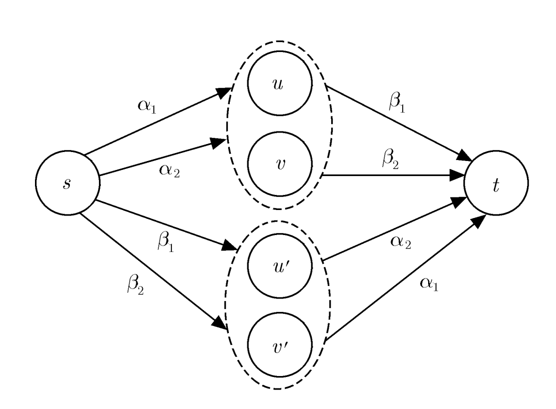



图 7.5 状态合并的束动作状态

由此可见，束动作 Kripke 结构中的状态数目比传统的 Kripke 结构要少. 如果需要利用传统的偏序约简算法实现模型检验，只需要将 Kripke 结构通过一定的方法转换成束动作 Kripke 结构，本书前面的叙述就是采用这种方法. 在模型检验前，在将 Kripke 结构转换成束动作 Kripke 结构过程中，首先需要确定束动作，确定束动作的具体步骤一般可以如下：

(1) 找出相互独立的动作；

(2) 找出从相同状态出发并发执行的动作，同时找出从不同状态出发并发执行回到相同状态的动作，判断是否构成相互独立的束动作，如果是，合并其中间的状态；

(3) 将单个相互独立的动作看成具有单个动作的独立束动作；

(4) 将其他剩余动作也看成具有单个动作的束动作.

按照上述步骤实现从动作到束动作的转换，需要启发性规则支持. 实际上，模型检验中的模型抽取本身可以直接从设计的系统模型得到束动作 Kripke 结构，相应地，需要验证的属性 LTL_x 公式也可以建立在束动作 Kripke 结构之上. 我们提出的束动作约简就是在传统的偏序约简处理前，进行必要的动作和状态合并，以减少状态空间. 具体步骤是：

(1) 在模型检验前，确定束动作；

(2) 根据需要检验的属性，确定 LTL_x 公式，以及 LTL_x 公式中用到的原子命题集合；

(3) 通过扫描迹等价建立束动作标记系统；

(4) 调用已有的传统的偏序约简算法进行模型检验.

为了形式化说明束动作 Kripke 结构如何从 Kripke 结构导出，必须在传统的 Kripke 结构中对束动作和束动作独立重新加以精确定义.

**定义 7.7.1**(束动作与束动作独立) 在 Kripke 结构 $\langle S, S_0, A, \rightarrow, L\rangle$ 中，设存在两个状态 $s, s' \in S$，并设存在两个不相交的由真并发动作组成的集合 $X$ 和 $Y$，其中 $X, Y \in \mathrm{enabled}(s)$. 同时满足下面两个条件：

(1) 存在一个状态集合 $T_s = \{t \mid t = \mathrm{succ}(s, \alpha), \alpha \in X\} \setminus \{s, s'\}$，使得动作集合 $Y \subseteq \{\beta \mid s' = \mathrm{succ}(t, \beta), t \in T_s\}$，并使得存在两个满射 $f : X \rightarrow T_s$ 和 $g : Y \rightarrow T_s$($Y$ 和 $X$ 是 $\mathrm{enabled}(s)$ 的非空子集，$f$ 表示从动作集合 $X$ 中的任何一个动作都是从状态 $s$ 出发使进程变迁到状态集合 $T_s$ 中的某一状态，并且状态集合 $T_s$ 中的任一状态至少由动作集合 $X$ 中的一个动作从状态 $s$ 出发变迁而来；$g$ 表示从动作集合 $Y$

中的任何一个动作都是从状态集合 $T_s$ 某一状态出发导致进程变迁到状态 $s'$，并且状态集合 $T_s$ 中的任一状态至少有动作集合 $Y$ 中的一个动作从该状态出发变迁到状态 $s'$)；

(2) 同时存在与条件 (1) 中状态集合不相交的另外一个状态集合 $R_s = \{r \mid r = \mathrm{succ}(s,\beta), \beta \in Y\} \setminus \{s, s'\}$，使得动作集合 $X \subseteq \{\alpha \mid s' = \mathrm{succ}(t,\alpha), t \in R_s\}$，并也存在两个满射 $f' : Y \to R_s$ 和 $g' : X \to R_s$，称动作组成的集合 $X$ 和 $Y$ 分别为**束动作**，并称 $X$ 和 $Y$ 是**相互独立的束动作**，记为 $X \sim_{\mathrm{ab}} Y$.

接下来定义状态合并的条件.

**定义 7.7.2**(状态可以合并) 在 Kripke 结构 $\langle S, S_0, A, \to, L\rangle$ 中，设存在一个状态集合 $T = \{t_1, t_2, \cdots, t_n\}$，其中 $t_i \in S (i = 1, 2, \cdots, n)$，如果存在一个原子命题集合 $\mathrm{AP}'$，使得 $L(t_1) \cap \mathrm{AP}' = L(t_2) \cap \mathrm{AP}' = \cdots = L(t_n) \cap \mathrm{AP}'$ 成立，称状态集合 $T$ 中的所有状态关于原子命题集合 $\mathrm{AP}'$ 可以合并成一个状态 (即选择其中一个状态就可以代表全部)，简称**状态可以合并**.

在一个 Kripke 结构 $\langle S, S_0, A, \to, L\rangle$ 中，根据定义 7.7.1，如果存在两个独立的束动作，并存在相应的两个状态集合 $T_s = \{t_1, t_2, \cdots, t_n\}$ 和 $R_s = \{r_1, r_2, \cdots, r_n\}$，其中 $t_i, r_i \in S$，如果也存在一个原子命题集合 $\mathrm{AP}'$，使得 $L(t_1) \cap \mathrm{AP}' = L(t_2) \cap \mathrm{AP}' = \cdots = L(t_n) \cap \mathrm{AP}'$ 和 $L(r_1) \cap \mathrm{AP}' = L(r_2) \cap \mathrm{AP}' = \cdots = L(r_n) \cap \mathrm{AP}'$ 成立，则状态集 $T_s$ 和 $R_s$ 中的状态也可以合并，即状态集合 $T_s$ 和 $R_s$ 中的状态可以分别合并成两个不同的状态，为了便于叙述，我们称这种情况为**关于独立束动作的状态可以合并**.

**定义 7.7.3**(束动作 Kripke 结构) 在 Kripke 结构 $\langle S, S_0, A, \to, L\rangle$ 中，按照定义 7.7.1 构建所有相互独立的束动作，并满足关于独立束动作的状态可以合并条件，如果将所有能合并的状态都合并，这样形成的 Kripke 结构，称为由 Kripke 结构导出的**束动作 Kripke 结构**.

这里的束动作 Kripke 结构与定义 7.4.1 的定义是一致的，只不过在这里是从传统的 Kripke 结构导出束动作 Kripke 结构来说明的.

**引理 7.7.1** 如果有一个 Kripke 结构 $M$ 和其导出的束动作 Kripke 结构 $M_{\mathrm{ab}}$，则 $M$ 和 $M_{\mathrm{ab}}$ 扫描迹等价.

**证明** 由于束动作 Kripke 结构 $M_{\mathrm{ab}}$ 是 Kripke 结构 $M$ 导出的，在 $M_{\mathrm{ab}}$ 中对每个束动作在变迁路径中的位置，与在 $M$ 中构成该束动作的动作所在的变迁路径中的位置对应排列.

根据定义 7.2.5 和定义 7.5.2，显然有：(1) 对 $M$ 中从初始状态开始的每条路径 $\sigma$，在 $M_{\mathrm{ab}}$ 中也存在一条从初始状态开始的路径 $\sigma_{\mathrm{ab}}$，满足 $\sigma \approx_{\mathrm{abst}} \sigma_{\mathrm{ab}}$；(2) 对 $M_{\mathrm{ab}}$ 中从初始状态开始的每条路径 $\sigma_{\mathrm{ab}}$，在 $M$ 中也存在一条从初始状态开始的路径 $\sigma$，满足 $\sigma_{\mathrm{ab}} \approx_{\mathrm{abst}} \sigma$. 所以 $M_{\mathrm{ab}}$ 和 $M$ 扫描迹等价.

由于上述状态合并不影响模型检验，所以从引理 7.7.1 直接可以得出下面定理.

**定理 7.7.1** 如果有一个 Kripke 结构 $M$，对任何给定的 LTL_x 公式 $\varphi$ 表示的进程属性，存在其导出的束动作 Kripke 结构 $M_{\mathrm{ab}}$，则有 $M \models \varphi$ 当且仅当 $M_{\mathrm{ab}} \models \varphi$.

**证明** 在一个 Kripke 结构 $M$ 中，设存在相应的原子命题集合 AP. 对任何给定的 LTL_x 公式 $\varphi$ 表示进程的属性，则存在相应的原子命题集合 AP′，其中 $\mathrm{AP}' \subseteq \mathrm{AP}$，用原子命题集合 AP′ 中的原子命题与操作算子 $\neg$, $\wedge$ 和 $U$(除 $X$(nXet) 外) 可以表示 LTL_x 公式 $\varphi$. 显然存在由 Kripke 结构 $M$ 导出的束动作 Kripke 结构 $M_{\mathrm{ab}}$，根据引理 7.7.1，则有 $M$ 和 $M_{\mathrm{ab}}$ 扫描迹等价，所以有 $M \models \varphi$ 当且仅当 $M_{\mathrm{ab}} \models \varphi$ 成立.

上述定理充分说明了从进程模型方面实现的基于束动作的偏序约简是正确的.

## 7.8 小　结

在并发系统形式化验证的模型检验技术中，偏序约简是一种缓解状态空间爆炸问题的有效方法. 其基本思想是，在迹理论的基础上通过动作独立关系和路径扫描迹等价关系的建立，使得对状态空间进行有选择地访问，达到减少访问状态的目的. 这种约简状态空间的方法以动作为基础，其粒度较小，对减少状态空间的作用有限. 我们研究了以束动作为基础的偏序约简方法，这里的束动作是由可以并发执行的动作构成的集合，其粒度较大，更能缩减状态空间，控制状态空间爆炸.

本章中基于束动作的偏序约简方法实际上是对传统的偏序约简方法的改进，包含传统的偏序约简方法. 我们从理论上说明了基于束动作的偏序约简的基本思想，并证明了这种方法在模型检验中的应用是正确的.

以下问题还需要进一步加以研究：(1) 如何从一般的 Kripke 结构中自动构造出束动作 Kripke 结构？(2) 如何利用束动作的思想直接改进传统的偏序约简算法？

# 参　考　文　献


[1] Aceto L. Action Refinement in Process Algebra [M]. Dept. of Computer Science. University of Sussex: Research Report, 1991.

[2] Aceto L and Hennessy M C B. Towards action-refinement in process algebras [J]. Information and Computation, 1993, 103: 204-269.

[3] Aceto L and Hennessy M C B. Adding action refinement to a finite process algebra [J]. Information and Computation, 1994, 115: 179-247.

[4] Alur R, Brayton R K, Henzinger T A, Qadeer S and Rajamani S K. Partial-order reduction in symbolic state-space exploration [C]. CAV'97, Lecture Notes in Computer Science, 1997, 1254: 340-351.

[5] Baeten J C M and Weijland W P. Process Algebra [M]. New York: Cambridge University Press, 1990.

[6] de Bakker J W and de Vink E P. Bisimulation semantics for concurrency with atomicity and action refinement [J]. Fundamenta Informaticae, 1994, 20: 3-34.

[7] de Bakker J W, de Roever W P, and Rozenberg G. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency [M]. Berlin: Springer, LNCS 354, 1989.

[8] Bekič H. Towards a Mathematical Theory of Processes [M]. Vienna: IBM Laboratory, Technical Report TR 25.125, 1971.

[9] Bergstra J A and Klop J W. Fixed Point Semantics in Process Algebra [M]. Amsterdam, Mathematical Centre: Tech. Report IW 208, 1982.

[10] Best E, Devillers R, Esparza J. General refinement and recursion operators for the Petri box calculus [C]. STACS'93, Lecture Notes in Computer Science, 1993, 665: 30-140.

[11] Bolognesi T and Brinksma E. Introduction to the ISO specification language LOTOS [J]. Computer Networks and ISDN Systems, 1987, 14(1): 25-59.

[12] Boudol G. Atomic actions [J]. Bull. Eur. Ass. Theoret. Comput.Sci., 1989, 38: 136-144.

[13] Bouldol G and Castellani I. Permutations of transitions: An Event Structure Semantics For CCS and SCCS [M]//de Bakker J W, de Roever W P and Rozenberg G. Linear Time, Branching Time and Partial Order in Logics and Models For Concurrency. Berlin: Springer, 1989, LNCS 354: 411-427.

[14] Boudol G, Castellani I. On the semantics of concurrency: Partial orders and transition systems [C]. TAPSOFT'87, Lecture Notes in Computer Science, 1987, 249: 123-137.

[15] Bryant R. Graph-based algorithms for Boolean functions manipulation [J]. IEEE Transactions on Computers, 1986, 35(8): 677-691.

[16] Burch J R, Clarke E M, McMillan K L. Dill D L, and Hwang L J. Symbolic model-checking:$10^{20}$ states and beyond [J]. Information and Computation, 1992, 98(2): 142-170.

[17] Busi N, van Glabbeek R J, Gorrieri R. Axiomatising ST bisimulation equivalence [C]. Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi, 1994, 169-188.

[18] Castellano L, De Michelis G, and Pomello L. Concurrency vs. interleaving: An instructive example [J]. Bull. Eur. Ass. Theoret. Comput. Sci., 1987, 31: 12-15.

[19] Cerry E, Corella F, Langevin M, Song X, Tahar S, and Zhou Z. Automated verification with abstract state machines using multiway decision graphs [C]. Formal Hardware Verification: Methods and Systems in Comparsion, Lecture Notes in Computer Science, 1997, 1287: 79-113.

[20] Clarke E M, Emerson E A and Sistla A P. Automatic verification of finite state concurrent systems using temporal logic specifications [J]. ACM Transactions on Programming Languages and Systems, 1986, 8(2): 244-263.

[21] Clarke E M, Enders R, Filkorn T and Jha S. Exploiting symmetry in temporal logic model Checking [J]. Formal Methods in System Design, 1996, 9: 77-104.

[22] Clarke E M, Grumberg O, Minea M and Peled D. State space reduction using partial order techniques [J]. STTT, 1999, 2(3): 279-287.

[23] Clarke E M, Grumberg O and Peled D. Model Checking [M]. Boston, MA: MIT Press, 1999.

[24] Czaja I, van Glabbeek R J and Goltz U. Interleaving semantics and action refinement with atomic choice [C]. Advances in Petri Nets, Lecture Notes in Computer Science, 1992, 609: 89-107.

[25] Darondeau P and Degano P. About semantic action refinement [J]. Fundamenta Informaticae, 1991, XIV: 221-234.

[26] Darondeau P and Degano P. Refinement of actions in event structures and causal trees [J]. Theoretical Computer Science, 1993, 118: 21-48.

[27] Darondeau P and Degano P. Causal trees [C]. Automata, Languages and Programming, Lecture Notes in Computer Science, 1989, 372: 234-248.

[28] Darondeau Ph, Degano P. Event structures, causal trees, and refinements [C]. Proc. MFCS'90, Lecture Notes in Computer Science, 1990, 452: 239-245.

[29] Degano P and Gorrieri R. Atomic refinement for process description languages [C]. MFCS'91, Lecture Notes in Computer Science, 1991, 520: 121-130.

[30] Degano P, De Nicola R and Montanari U. Partial Orderings Descriptions and Observations of Nondeterministic Concurrent Processes [M]//de Bakker J W, de Roever W P and Rozenberg G. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. Berlin: Springer, 1989, LNCS 354: 438-466.

[31] Degano P, Gorrieri R. A causal operational semantics of action refinement [J]. Information and Computation, 1995, 122: 97-119.

[32] Emerson E A and Sistla A P. Symmetry and model checking [C]. Proceedings of the 5th Workshop on Computer-Aided Verification, 1993, 463-478.

[33] Fecher H, Majster-Cederbaum M and Wu J. Action refinement for probabilistic processes with true concurrency models [C]. Lecture Notes in Computer Science, 2002, 2399: 77-94.

[34] Gabbay D, Pnueli A, Shelah S and Stavi J. On the temporal analysis of fairness [C]. Conference Record of the 7th ACM Symposium on Principles of Programming Languages. ACM Press, 1980, 163-173.

[35] van Glabbeek R J and Goltz U. Equivalence notions for concurrent systems and refinement of actions [C]. Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, 1989, 379: 237-248.

[36] van Glabbeek R J. The linear time-branching time spectrum [C]. CONCUR'90, Lecture Notes in Computer Science, 1990, 458: 298-297.

[37] van Glabbeek R J and Goltz U. Refinement of actions in causality based models [C]. Stepwise Refinement of Distributed Systems- Models, Formalisms, Correctness, Lecture Notes in Computer Science, 1990, 430: 267-300.

[38] van Glabbeek R J, Goltz U. A Deadlock-sensitive Congruence for Action Refinement [M]. Institut fuer Informatik, Technische Universitaet Munchen: SFB-Bericht 342/23/90 A, 1990.

[39] van Glabbeek R J, Weijland W P. Branching time and abstraction in bisimulation semantics [J]. J. ACM, 1996, 43(3): 555-600.

[40] van Glabbeek R J and Goltz U. Refinement of Actions and Equivalence Notions for Concurrent Systems [M]. University of Hildesheim: Hildesheimer Informatik-Bericht 6/98, 1998.

[41] van Glabbeek R J, Goltz U. Refinement of actions and equivalence notions for concurrent systems [J]. Acta Informatica, 2001, 37(4/5): 229-327.

[42] Godefroid P. Partial Order Models for the Verificaton of Concurrent Systems—An Approach to the State Explosion Problem [M]. Universite de Liege: PhD thesis, 1994.

[43] Godefroid P and Wolper P. A partial approach to model checking [C]. Proceeding of the Sixth Annual IEEE Symposium on Logic in Computer Science. IEEE Society Press, 1991, 406-415.

[44] Godefroid P. Partial-order Methods for the Verification of Concurrent Systems: An Approach to The State-explosion Problem [M]. Berlin: Springer, LNCS 1032, 1996.

[45] Goltz U and Wehrheim H. Modelling causality by dependency of actions in branching time semantics [J]. Information Processing Letters, 1996, 59(4): 179-184.

[46] Gorrieri R, Marchetti S and Montanari U. A2CSS: Atomic actions for CCS [J]. Theoretical Computer Science, 1990, 72: 203-223.

[47] Gorrieri R. A hierarchy of system descriptions via atomic linear refinement [J]. Fundamenta Informaticae, 1992, 16: 289-336.

[48] Gorrieri R and Montanari U. On the implementation of concurrent calculi in net calculi: Two case studies [J]. Theoretical Computer Science, 1995, 141: 195-252.

[49] Gorrieri R, Rensink A. Action Refinement [M]//Bergstra J A, Ponse A and Smolka S A, editors. Handbook of Process Algebra. New York: Elsevier Science, 2001: 1047-1147.

[50] Hermanns H and Ribaudo M. Exploiting symmetries in stochastic process algebras [C]. Proc. of European Simulation Multiconference, SCS Europe, 1998, 763-770.

[51] Hoare C A R. Communicating Sequential Processes [M]. Upper Saddle River: Prentice-Hall, 1985.

[52] Huhn M. Action refinement and property inheritance in systems of sequential agents [C]. Concur'96, Lecture Notes in Computer Science, 1996, 1119: 639-654.

[53] Jiang J, Wu J. The preservation of interleaving equivalences [C]. Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems. IEEE Computer Society Press, 2005, 580-589.

[54] Jiang J, Wu J. Symmetry and autobisimulation [C]. Proceedings of the 6th International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE Computer Society Press, 2005, 866-870.

[55] Jiang J, Wu J and Yan W. Structural reductions in process algebra languages [C]. Proceedings of the 11th Joint International Computer Conference. World Scientific Publishing Co., 2005, 596-600.

[56] Jategaonkar L, Meyer A R. Self-synchronization of concurrent processes [C]. Eight Annual IEEE Symposium on Logic in Computer Science. IEEE Computer Society Press, 1993, 409-417.

[57] Jategaonkar L and Meyer A R. Testing equivalences for Petri nets with action refinement [C]. Concur'92, Lecture Notes in Computer Science, 1992, 630: 17-31.

[58] Keller R M. Formal verification of parallel programs [J]. Communications of the ACM, 1976, 19(7): 371-384.

[59] Knijnenburg P M W and Kok J N. The semantics of the combination of atomized statements and parallel choice [J]. Formal Aspects of Computing, 1997, 9(5/6): 518-536.

[60] Lien Y E. Study of theoretical and practical aspects of transition systems [M]. University of California, Berkeley. PhD Disseration, 1972.

[61] Loogen R, Goltz U. Modelling nondeterministic concurrent processes with event structures [J]. Fundamenta Informaticae, 1991, 14: 39-74.

[62] Fecher H, Majster-Cederbaum M and Wu J. Refinement of actions in a real-time process algebra with a true concurrency model [J]. Electronic Notes in Theoretical computer Science, 2002, 70(3): 620-640.

[63] Majster-Cederbaum M, Wu J. Action refinement for true concurrent real time [C]. Proc. ICECCS'01, IEEE Computer Society Press, 2001, 58-68.

[64] Majster-Cederbaum M, Wu J. Towards action refinement for true concurrent real time [J]. Acta Informatica, 2003, 39: 1-47.

[65] Majster-Cederbaum M, Wu J. Adding action refinement to stochastic true concurrency models [C]. ICFEM'03, Lecture Notes in Computer Science, 2003, 2885: 226-245.

[66] Majster-Cederbaum M, Wu J, Yue H. Refinement of actions for real-time concurrent systems with causal ambiguity [J]. Acta Informatica, 2006, 42(6/7): 389-418.

[67] Mazurkiewicz A. Basic notions of trace theory [C]. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency, Lecture Notes in Computer Science, 1989, 354: 285-363.

[68] Meyer A. Observing truly concurrent processes [C]. Theoretical Aspects of Computer Software, Lecture Notes in Computer Science, 1994, 789: 886-898.

[69] Meyer A. Concurrent process equivalences: Some decision problems [C]. STACS'95, Lecture Notes in Computer Science, 1995, 900: 349-362.

[70] Milner R. A Calculus of Communicating Systems [M]. New York: Springer, 1980.

[71] Nielsen M, Plotkin G D and Winskel G. Petri nets, event structure and domains, Part I [J]. Theoretical Computer Science, 1981, 13(1): 85-108.

[72] Nielsen M, Engberg U, Larsen K S. Fully Abstract Models for a Process Language With Refinement [M]//de Bakker J W, de Roever W P and Rozenberg G. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. Berlin: Springer, 1989, LNCS 354: 523-549.

[73] Park D. Concurrency and automata on infinite sequences [C]. Theoretical Computer Science: 5th GI-Conference, LNCS 104, 1981.

[74] Pratt V R. Modeling concurrency with partial orders [J]. International Journal of Parallel Programming, 1986, 15(1): 33-71.

[75] Peled D. Combining partial order reductions with on-the-fly model checking [C]. CAV'94, Lecture Notes in Computer Science, 1994, 818: 377-390.

[76] Peled D, Wilke T. Stutter-invariant temporal properties are expressible without the next-time operator [J]. Information Processing Letters, 1997, 63(5): 243-246.

[77] Penczek W. Model-checking for a subclass of event structures [C]. TACAS'97, Lecture Notes in Computer Science, 1997, 1217: 145-164.

[78] Plotkin G D. A Structural Approach to Operational Semantics [M]. Computer Science Department, Aarhus University: Technical Report DAIMI FN-19, 1981.

[79] Pomello L. Some equivalence notions for concurrent systems— an overview [C]. Advances in Petri Nets, Lecture Notes in Computer Science, 1985, 222: 381-400.

[80] Rabinovich A and Trakhtenbrot B A. Behaviour structure and nets [J]. Fundamenta Informaticae, 1988, XI(4): 357-404.

[81] Rensink A. Models and Methods for Action Refinement [M]. University of Twente, Enschede, Netherlands: PhD thesis, 1993.

[82] Rensink A. An event-based SOS for a language with refinement [C]. Structures in Concurrency Theory, Workshops in Computing. Springer, 1995, 294-309.

[83] Rotman J J. An Introduction to The Theory of Groups [M]. New York: Springer-Verlag, 1994.

[84] Starke P H. Reachability analysis of Petri nets using symmetries [J]. Systems Analysis Modelling and Simulation, 1991, 8: 293-303.

[85] Sun X, Zhang W, Wu J. Event-based operational semantics and a consistency result for real-time concurrent processes with action refinement [J]. Journal of Computer Science and Technology, 2004,19(6): 828-840.

[86] Tretmans J. Conformance testing with labelled transition systems: Implementation relations and test generation [J]. Computer Networks and ISDN Systems, 1996, 29: 49-79.

[87] Valmari A. A stubborn attack on state explosion [C]. Proc. the Second Workshop on Computer-Aided Verification, Lecture Note in Computer Science, 1990, 531: 156-165.

[88] Vogler W. Bisimulation and action refinement [C]. STACS'91, Lecture Note in Computer Science, 1991, 480: 309-321.

[89] Vogler W. Failure semantics based on interval semiwords is a congruence for refinement [J]. Distributed Computing, 1991, (4): 139-162.

[90] Vogler W. Modular construction and partial order semantics of Petri nets [C]. Lecture Note in Computer Science, 1992, 625: 625-648.

[91] Vogler W. The limit of $\mathrm{Split}_{\mathrm{n}}$-language equivalence [J]. Information and Computation, 1996, 127(1): 41-61.

[92] Winskel G. Event structure semantics for CCS and related languages [C]. ICALP'82, Lecture Note in Computer Science, 1982, 140: 325-392.

[93] Winskel G. Event structures [C]. Petri Nets: Applications and Relationships to Other Models of Concurrency, Lecture Note in Computer Science, 1987, 255: 325-392.

[94] Winskel G. An Introduction to Event Structures [M]// de Bakker J W, de Roever W P and Rozenberg G. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. Berlin: Springer, 1989, LNCS 354: 364-397.

[95] Wirth N. Program development by stepwise refinement [J]. Commun. ACM, 1971, 14(4): 221-227.

[96] Wu J. Action refinement in timed LOTOS [C]. Proc. of ASCM'01, World Scientific Publ., 2001, 183-192.

[97] Wu J. Logic programming—-taking advantage of symmetry [C]. Proc. of ASCM'00, World Scientific Publ., 2000, 100-109.

[98] Wu J, Fecher H. Symmetric structure in logic programming [J]. Journal of Computer Science and Technology, 2004, 19(6): 803-811.

[99] Weyl H. Symmetry [M]. Princeton: Princeton U. Press, 1952.